ARCADE BOARD

OPERATING MANUAL

THIRD MILLENNIUM ENGINEERING CORPORATION

1015 GAYLEY AVENUE SUITE 394

LOS ANGELES, CA 90024

COPYRIGHT

DISCLAIMER

Third Millennium Engineering Corporation makes no representations or warranties with respect to the contents hereof, to the Arcade Board and to the Amparcade language, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Third Millennium Engineering Corporation reserves the right to revise this publication and to make changes from time to time in the content hereof, and to change the design and specifications of the Arcade Board and of the Amparcade language, without obligation of Third Millennium Engineering Corporation to notify any person of such revisions or changes.

TRADEMARKS

IMPORTANT NOTICE

Various figures and tables in this manual were reprinted with the kind permission of Texas Instruments Incorporated.

First Edition: October 1983

LIMITED 90 DAY WARRANTY

Third Millennium Engineering Corporation warrants this product against defects in materials and workmanship for a period of 90 days from the date of purchase. We will repair or replace products that are proven to be defective during the warranty period, provided that they are returned to Third Millennium Engineering Corporation at the customer's expense. No other warranty is expressed or implied. We reserve the right to refuse repairs on any product that we judge to have been subjected to abnormal electrical or mechanical abuse. Products out of warranty will be repaired for a nominal fee covering the costs of labor and materials. Before sending your Third Millennium Engineering Corporation unit in for repair, contact our Customer Service Representative for a Return Authorization Number. Call (213) 473-2102

The above warranties for goods are in lieu of all warranties, expressed, implied or statutory, including, but not limited to, any implied warranties of merchantability and fitness for a particular purpose, and of any other warranty obligations on the part of Third Millennium Engineering Corporation. In no event shall Third Millennium Engineering Corporation or anyone else involved in the development and manufacture the this product be liable for indirect, consequential or special damages, such as, but not limited to, loss of anticipated profits or other benefits resulting from the use of this product, or arising from any breach of this warranty. As some states do not allow the exclusion or limitation of consequential damages, the above limitation may not apply to you.

- -

Cut at dotted line and mail to:

Third Millennium Engineering Corporation
1015 Gayley Ave, Suite 394
Los Angeles, CA 90024

Name ________________________________ Date ______________

Address ______________________________ Phone _____________

City ____________________ State _____ Zip Code ____________

Serial Number _______________ Product _____________________

* *

ARCADE BOARD OPERATING MANUAL

TABLE OF CONTENTS

* *

ARCADE BOARD AMPARCADE COMMAND DOCUMENTATION

TMS 9918A VIDEO DISPLAY PROCESSOR REFERENCE MANUAL

AY-3-8910 PROGRAMMABLE SOUND GENERATOR REFERENCE MANUAL

INTRODUCTION

Congratulations on your purchase of the Arcade Board! You now have in your possession a peripheral that redresses the woeful state of the Apple II's graphics and sound generating capabilities which are major drawbacks of the Apple II when compared to the host of newer and less expensive home computers now on the market. No longer need you be envious of the Atari 400/800, the Commodore 64, the VIC 20, the TI 99/4A, etc. With the Arcade Board, you can boast of having the best graphics and sound generating computer around, as well as having a computer which is capable of serious computing.

The Arcade Board uses special hardware that gives the Apple II, II+, and IIe the same powerful color graphics, animation, sound and music capabilities as the coin-operated arcade machines. The Arcade Board uses the same chips employed by many of the major arcade machine manufacturers - namely the Texas Instruments TMS 9918A Video Display Processor (VDP) and the General Instruments AY-3-8910 Programmable Sound Generator (PSG).

These special chips off-load the Apple's 6502 microprocessor (or the Z-80 if you are using the Microsoft Z-80 SoftCard) from all the labor normally involved in creating animated color graphics and sound effects. What's more, these chips do a far better job of it - so good, in fact, that the Apple can honestly be said to possess true arcade-quality graphics and sound when used with the Arcade Board. For example, to move an object, say a blue ball of about 32 pixels wide, the 6502 would have to erase the ball at the old location on the screen and redraw it at the new location. This operation could involve thousands of 6502 machine instructions. In order to have fast, smooth animation, you would definitely have to use assembly or machine language, since BASIC would be far too slow. However, with the Arcade Board, this same blue ball could be moved to any location on the screen with from four to ten (depending on conditions) 6502 machine instructions. This is so fast that even Applesoft or Integer BASIC could handle the job. A second example of the ARCADE BOARD'S capabilities is the creation of sound effects. Normally the Apple toggles a port to move the speaker cone in or out. The rate at which this is done determines the frequency of the tone. To have a continuous tone the 6502 must continuously toggle the speaker port: While doing so, it can not do anything else, such as move objects around on the screen. This is the reason so many games for the Apple II have so few sound effects. You can have animation or sound but not both at the same time (other than a few rudimentary clicks). You may have noticed the result of this conflict in games where there is the sound of an explosion and all movement on the screen freezes until the sound is over. Using the Arcade Board all you (or the 6502) has to do to get a sound effect is tell the PSG what to do. The PSG will automatically and continually create the desired sound until told to stop. Meanwhile the 6502 can go on to do other things, such as animation. The combination of both the TMS 9918A VDP and the AY-3-8910 PSG off-loads the 6502 of most of the dirty work involved in the animation of color graphics and sound effects. Using only the BASIC interpreter, spectacular color graphics and sound effects can be created. Now the average BASIC programmer no longer needs to purchase expensive machine language programs in order to produce fancy animated color graphics with sound effects or music. With BASIC alone you can create your own spectacular programs that far surpass in quality all the machine language programs that use only the standard Apple II graphics and sound capabilities. We are certain that you will enjoy your Apple more than ever before now that you can create fancy color graphics and sound effects using the Arcade Board -- After all, that's what you bought your Apple for in the first place, isn't it?

INSTALLATION OF THE ARCADE BOARD

CHECK LIST

You should have received the following items with your purchase of the Arcade Board:

- one Arcade Board printed-circuit board
- one Demonstration diskette
- one PRELIMINARY instruction manual
- one Warranty Card to send in for your 90 day warranty and to register for your new complete manual free of charge when it becomes available.

In addition you will need the following items for installation:

- an Apple II, II+, or IIe computer
- one disk drive with disk controller card
- a composite video color monitor or a color TV with RF modulator
- at least one cable with male RCA phono jacks on both ends. If you have a color monitor with a sound input, you will need another cable with a male RCA phono jack on one end and the correct connector to fit into your monitor's sound input jack on the other end. For the NEC and BMC color monitors, this is a male RCA phono jack; but for the Amdek Color I monitor, it is a standard earphone jack.
- a small slotted head screwdriver

Before you use the Arcade Board you will have to install it in slot 4 - READ EITHER "COLOR TV USAGE" OR "COLOR MONITOR USAGE" BELOW BEFORE ACTUALLY INSERTING THE ARCADE BOARD INTO ANY SLOT - You will need to "tune" the Arcade Board's colors so that they will be correct for your particular color TV or color monitor. After this it would be a good idea to run the supplied test programs and demonstration programs just to make sure everything is installed and working correctly. Then go on to do your own programming.

COLOR TV USAGE

If you have a color TV, you will need an RF modulator such as the Sup-R-Mod II RF modulator. On the left edge of the Arcade Board is a 4-pin connector similar to one on the Apple's motherboard to which the RF modulator is normally attached. You will need to disconnect the RF modulator from the Apple motherboard and connect it to this 4-pin connector on the Arcade Board.

CAUTION: BE CAREFUL! THE BLACK WIRE COMING OUT OF THE RF MODULATOR CONNECTOR SHOULD BE ALIGNED WITH THE TOP PIN OF THE 4-PIN CONNECTOR ON THE ARCADE BOARD - THIS PIN IS MARKED "GND" FOR GROUND. MAKE SURE THIS PIN IS CONNECTED TO THE BLACK WIRE ON THE RF MODULATOR. IF YOU SHOULD REVERSE THE ORIENTATION OF THE THE CONNECTOR YOU MAY DAMAGE THE RF MODULATOR OR THE ARCADE BOARD, ALTHOUGH THIS IS NOT LIKELY.

The Rf modulator is used as it is normally except now it is connected to the Arcade Board.

Gently insert the Arcade Board into the slot from which you choose to operate it. We recommend either slot 4 or 7, since most of the software on the demo disk is designed for slot 4. Slot 7 is rarely used by other peripherals and would be ideal for the Arcade Board. Fitting the board into the slot you select should not be a problem since the boards have been checked and tested at the factory. If you find the Arcade Board does not easily fit into the desired slot, it may be because the width of the edge-connector portion (where the gold plated contacts are) is a few thousandths of an inch too wide. This area should be about 3.6 inches. A small file should be used to file down the sides of this area to make it fit in easier.

Take the tubular connector on the end of the single wire coming out of the Arcade Board. Insert it into the hole in the video out female RCA plug on the outside of the Apple near the cassette in and out plugs. This wire feeds the normal Apple video signal into the Arcade Board for soft-switching between normal Apple video and Arcade Board video. Since the RF modulator is no longer connected to the Apple motherboard, there must be a way to get the normal video signal to the RF modulator. The wire connected to the video out plug serves this purpose.

The bottom female RCA phono connector on the Arcade Board is for the sound output. Since you are using an RF modulator designed for use with the Apple II, which, unlike many other computers, uses its own on-board speaker, you will need either an external amplifier and speaker or some kind of RF modulator that also transmits audio signals to the TV. You can use a normal stereo amplifier or you can buy a cheap amplifier (e.g. from Radio Shack for under $15) to amplify the sound output. Since it will depend on the type you get, we cannot supply instructions on hooking up to a special RF modulator.

COLOR MONITOR USAGE

To hook up the Arcade Board to a color monitor you will need to disconnect the video input cable of your color monitor from the Apple's video output plug (located at the rear of the Apple), and connect it to the top RCA connector of the Arcade Board. If your color monitor has an audio input jack then you will need a cable with a male RCA phono connector on one end and the appropriate connector for your particular color monitor's audio input jack on the other end. The bottom RCA phono connector of the Arcade Board is the audio output. Connect this to your monitor's audio input. Gently insert the Arcade Board into whatever slot you choose to operate it from (except slot 0). We recommend slot 4, since one of the programs (MLDEMO) on the demo disk will work only in slot 4, due to the lack of a slot finder routine. Insert the tubular connector on the end of the single wire coming out of the Arcade Board into the Apple's video output plug. (See paragragh 3 under Color TV Useage above.)

FOR USERS OF ONLY ONE MONITOR OR TV

Once the Arcade Board is installed, you should be ready to boot up your Apple. Since D.O.S. 3.3 is a product of Apple Computer, Inc. we can not put D.O.S. 3.3 on the demo disk. Therefore you will have to boot up on another disk which has D.O.S. 3.3 on it. After booting, your screen will be blank as the initial state of the Video Display Processor (VDP) used in the Arcade Board is to disable the external video signal (the Apple II's video output) from passing through to the monitor or TV. The HELLO program supplied on the demo disk is a short program which tells the VDP to allow the external video signal to pass through to your monitor or TV. Once this HELLO program has been run, you should be able to see the normal Apple II's video signal with the blinking cursor. We recommend that you copy the demo disk onto a disk that has D.O.S. 3.3 on it. The HELLO program on this copy should be the HELLO program supplied on the demo disk. If you boot up with this disk, then the Apple's video would be automatically enabled when the HELLO program is run.

In addition to enabling the Apple's video, the HELLO program pokes a short machine language program into memory starting at address $300. Whenever the RESET key is pressed, this routine is executed and Apple video is enabled. Any program that changes the reset vector (Autostart ROM only), will disable this feature. This feature will not work on Apples that have the Old Monitor ROM, since the reset vector is not in RAM, and thus can not be changed to point to a user program.

Some disks require you to boot up on that disk in order to run the programs on the disk. This is true of protected disks. In order to run these programs, you will first have to boot up on a disk that has the special HELLO program on it which enables the Apple's video. Then you can boot up on the protected disk, using a PR#6 command. If the program on the protected disk changes the reset vector to point back to the start of the protected program (as many of them do), then hitting RESET will make the screen go blank.

The HELLO program may be integrated into other HELLO programs, such as the HELLO program which loads a 16K RAM card, if you have one. This is easily done by placing the supplied HELLO program at the end of any other HELLO program. It should be placed at the end, because the HELLO program changes the reset vector at locations $3F2 and $3F3 (decimal addresses 1010 and 1011) to point to the short machine language program which enables the Apple's video. Whenever you hit RESET, this program should run. Some HELLO programs change the reset vector, that is why the supplied HELLO program should be placed at the end of other HELLO programs. At the end of the short machine language program which re-enables the Apple's video output, control is passed to the program which was originally pointed to by the original reset vector, before it was overwritten. The reset vector normally points to the entry point to D.O.S, so after re-enabling the Apple's video output, control is passed to D.O.S.

The Apple's video output level may have to be adjusted to get the right level of brightness. If you have never used your Apple with a monitor before the video output level may not even be set so that you could see anything on the screen. In the upper right hand corner of the Apple II motherboard there is a small round black object with a cross shapped depression made to fit a screw driver. Refer to page 10 of the Apple II Reference Manual for a picture; it is listed as the Level Adjustment Potentiometer. You should turn this dial until the level of brightness of the Apple video signal is suitable. In some cases when this dial is turned down too far, the screen goes totally white.

TUNING

By now you should have the Arcade Board installed and hooked up to your color monitor or color TV. Run a simple BASIC program like TILES. If the output is in black and white, then you will need to tune the Arcade Board. In some cases even if the output is colored, tuning can make it better. To tune the Arcade Board you will need to turn the small trimmer capacitor located in the upper right-hand corner of the Arcade Board. Turn this with a small screw-driver until the output is suitable. It would be easier to do this if the disk controller were in slot 1, 2, or 3 and the Arcade Board were in slot 4.

TEST PROGRAMS

Included on the demo disk are some test programs. Run VDPTEST to test the VDP. If it runs for several minutes without a beep, then the VDP is OK. Then run PSGTEST, you should hear all kinds of sounds. If you do not hear any sound, then the PSG is not working.

* * * * * * * TROUBLE SHOOTING * * * * * * *

BLACK AND WHITE IMAGE, BUT NO COLOR

COLOR TV USAGE: If you are running one of the Arcade Board color graphics demo programs and all you can see is black and white images on your TV set, but no color, then you have a tuning problem. This is tricky to correct, because there are three things that all must be tuned correctly: the color TV tuning control (turn off any auto-color controls that your TV may have), the RF modulator, and the Arcade Board. If any two of these are tuned correctly, but not the third, you may never know it, because you will see only black and white.

The easiest way is to tune your color TV and RF modulator separately with a normal Apple color graphics program, such as the Color Demo program which comes on the D.O.S. 3.3 System Master diskette. If your RF modulator and color TV were adjusted already before you bought the Arcade Board, and you have not changed their settings since, then you may move on to the next paragraph. But, if in attempting to tune the Arcade Board, you have changed the setting of either your color TV, RF modulator, or both, then you will have to readjust them back to the settings that worked before. Turn off the power to your Apple and remove the Arcade Board and all connections to it, and then hook up the RF modulator to the Apple's motherboard as you have always done before. Then run some program that uses normal Apple color graphics, and adjust both the RF modulator and the color TV so that the colors come out alright. Once this is done turn off the power and hook up the Arcade Board again according to the instructions in the section 'COLOR TV USAGE'.

Since you are using a color TV, it is assumed that you may have never adjusted the level of the video signal which comes out of the Apple's monitor output RCA connector at the back of the Apple. On Apple II's and II+'s there is a small potentiometer in the upper right hand corner of the motherboard (Apple IIe owners need not worry about this step). Do not confuse this with the color trim capacitor, also in the same general area. The monitor video output level must be adjusted so that its output voltage is roughly equal to that of the Aracade Board's TMS9918A video output voltage, otherwise the Apple's text and graphics may seem dimmer or brighter than the Arcade Board's.

Boot up a disk with D.O.S. 3.3 on it and run the special HELLO program to enable Apple video. Get some text on the screen; LISTing the HELLO program is easiest. Now turn the video output level adjustment dial with a small screw driver so that the text can be seen clearly without being too dim, or too bright. If the dial is turned down too low the text will be weak and pale, if turned up too high the letters will be very bright and may even 'smear'. A good setting is only as high as is needed to see the letters clearly without being too dim.

Now run an Arcade Board color demo program such as TILES. If you do not see the output in color, then there are only two things that may have to be adjusted. Try turning the trimmer capacitor on the Arcade Board around until you can see color. If you still can not get a color picture, then turn the setting of the RF modulator down a bit (counter clockwise on the Sup-R-Mod II) and try again. Or if the picture looks very dim, turn the setting of the RF modulator up and try again. You should be able to get color eventually by fiddling around in this manner.

Sometimes while turning the trimmer capacitor, you may get color, but as soon as you take your hand and screwdriver away it reverts back to black and white. This is most noticeable with metal handled screwdrivers and is due to the small capacitance of your hand. You will have to fiddle around until you can get the color to come out right.

If you still have problems with getting color, then call us at (213) 473-2102 and we will try to help.

COLOR MONITOR USAGE: You should have little problem tuning the Arcade Board with a color monitor. If the regular Apple's graphics, without the Arcade Board installed, show up in color on your monitor, then it is only a simple matter of tuning the trimmer capacitor on the Arcade Board until you get color. If you can not get color at all then call us for help.

DARKER COLORED VERTICAL BAR(S) MOVING ACROSS THE SCREEN

If you have managed to get a color picture from the Arcade Board on your color monitor or color TV, but you see one or more vertical bars or bands, whose color is slightly darker than the screen color, moving across the screen, then you have the Apple's video output level adjusted too high. To see this, unplug the black wire connected to the Apple's monitor output jack at the back of the Apple. The bars should go away. The Apple's video signal is bleeding into the video signal of the TMS9918A chip and causing interference.

This effect is eliminated or reduced to negligible levels by adjusting the Apple's video output potentiometer (located in the upper right hand corner of the Apple motherboard) to the point where you do not see any bars. Do not turn this potentiometer down too far or else, the regular Apple video signal will be too weak to see on the screen when Apple video is enabled. You should find a good compromise adjustment level that allows the regular Apple video signal to be clear, but without these bars when in Arcade Board mode.

Also it is possible to slightly turn the trimmer capacitor on the Arcade Board one way or the other to eliminate or reduce these bars from the TV screen. If you can not totally eliminate them by adjusting the Apple's video output level, then try doing this, taking care not to loose color. Usually a combination of both adjustments will get rid of these bars. Of course if you have two monitors of TV's then you will not have this problem at all.

NO PICTURE AT ALL

SCREEN ALL WHITE: If the screen is all white, whether in Arcade Board video mode or Apple video mode, then either the video amplifier output transistor on the Arcade Board is burned out, or the TMS9918A chip is defective. As the most likely cause of this problem is a burnt out transistor, you may try replacing the output transistor (use 2N3904, 2N4401, or equivalent), and if you do not butcher the PC board, this will not void your warranty. If this does not work, then most likely the video chip is bad. You will have to send the board in for repair, unless you wish to replace the video chip yourself. The TMS9918A can be bought from most Texas Instruments distributors for about $40.00 each.

If the Arcade Board video works, but in Apple mode the screen is all white, then this is usually caused by the video output level potentiometer on the Apple's motherboard being turned down too low. Or else there may be a bad connection, so that the Apple's video signal does not get to the Arcade Board. Try turning up the video output potentiometer.

SCREEN ALL BLACK: This is most likly due to a bad TMS9918A chip. You will have to have it replaced by us or someone else. There is also the possibility that the video output transistor on the Arcade Board is bad.

RIGHT EDGES OF IMAGES SMEARED OUT

When using an RF modulator and the image on the screen is smeared out along the right edges of the objects, then the RF modulator is turned up too high. This is usually accompanied by no color in the Arcade Board generated graphics. This is easily corrected by turning down the adjustment of the RF modulator.

GARBAGE ON THE SCREEN

When running some of the demo programs and you see junk on the screen that you think should not be there, then most likely there is some bad RAM on the Arcade Board. Run VDPTEST for several minutes. If you hear a beep, then you most likely have bad RAM. There is also the possibility that the TMS9918A chip could be defective as well. You could try replacing the RAM chips on the Arcade Board with some other 200 nanosecond access time 4116 dynamic RAM chips to test this out.

When running your own programs and you see junk on the screen, check your program logic, especially whether you have disabled all unused sprites, before you check the hardware.

NO SOUND

If you can not get any sound, first check your connections to your amplifier or monitor with an audio input. If these are all right, then you may have a bad sound chip. But the two 4052 chips or the two 747's may also be defective. These are cheap and readily available, if you wish to try replacing them

BAD SOUND

If you think that your sound generator is sometimes not putting out the right sounds, then you may have a bad sound chip. This would be a random problem, where sometimes you get a sour note while running the same program. You can test this by running a program that plays the same note over and over. If every now and then you hear a different note, then you have a defective sound chip. You can order one from us for $10.00 or if under warranty, we will repair your board for you.

NO SOUND AND NO GRAPHICS

This is most likely due to the interface chips being defective. Try replacing the 74LS00 (or 74C00), the 74LS04, and the 74LS42 chips. These are readily available and very cheap.

APPLE DOES NOT WORK WHEN ARCADE BOARD INSTALLED

This problem can be caused by a number of reasons. Usually it could be a burnt out 10 micro Farad tantalum capacitor on the Arcade Board. These capacitors are rated for 16 volt operation, so they should not burn out, but sometimes a defective one might. If this happens the Apple's power supply will see a short circuit and shut off automatically, preventing damage to the Apple itself. You can usually see the orange capacitors discolored if they are burnt out.

If you have replaced any chips yourself, you may have inserted them backwards or bent a pin, etc. This could also cause the Apple's power supply to shut off. There is little possibility of a defective Arcade Board from causing any permanent damage to your Apple, as the power supply just shuts off.

===

ARCADE BOARD .

AMPARCADE TUTORIAL MANUAL

===

1.0 INTRODUCTION

The Amparcade Graphics and Sound language for the Arcade Board is a set of over 60 BASIC-like command extensions to Applesoft BASIC, which allows the average programmer to take advantage of the powerful and flexible features of the Arcade Board, without having to spend many laborious hours studying the technical details of the TMS 9918A Video Display Processor (VDP) and the AY-3-8910 Programmable Sound Generator (PSG). The Amparcade language allows you to get started right away, writing your own programs featurnig arcade-quality animation, with 16 color, high resolution graphics and simultaneous sound effects and music.

As you get more familiar with the Arcade Board's capabilities, you may want to learn more about the technical details. The VDP Reference Manual and the PSG Reference Manual provide all the technical details necessary in order to take full advantage of the real power of the Arcade Board. Most of you will find, however, that the Amparcade language provides more than enough power to fully enjoy your Arcade Board, and utilize its capabilities to your complete satisfaction and enjoyment. You will find that the Arcade Board and the Amparcade language sky-rockets your Apple II's graphics and sound capabilities to a whole new level of excitement, never imagined before. You will soon be able to make your Apple II do the things that you always wished it could do. The Amparcade language allows you to program the Arcade Board with the ease and convenience of BASIC, but with the execution speed of machine language, and the spectacular graphics and sound effects of the commercial coin-operated video game machines.

The Amparcade commands use the ampersand (&) hook of Applesoft BASIC, and therefore require you to type an ampersand ' & ' sign in from of every Amparcade command. In addition a comma must be typed before every variable or expression, even if it is the first one after the command name. For example: & TNA, P requires that the comma be there between & TNA and P. If not a SYNTAX ERROR will result.

2.0 GETTING STARTED

At this point you should have successfully installed, tuned and tested your Arcade Board. If not go back and read the sections on installation, testing and troubleshooting before continuing any further. You should also have made a backup copy of your demonstration disk, especially of the file 'AMPARCADE' which is the Amparcade interpreter.

The first thing to do is to load in the Amparcade language interpreter into your Apple's memory. The Amparcade interpreter resides just under D.O.S. 3.3 and starts at address 34816, or in hexadecimal notation $8800. Therefore your Apple must have at least 48 K of RAM and also have Applesoft in ROM or on a 16 K RAM card. Insert the demo disk and turn on the power to boot up. After the Third Millennium greeting messages appear on the screen, type:

```
BRUN AMPARCADE
```

followed by a carriage return. We will assume from now on that you know that you have to hit RETURN after every command you type, so we will not mention it again.

You should see the message:

```
ARCADE BOARD AMPARCADE INTERPRETER

COPYRIGHT (C) 1983

THIRD MILLENNIUM ENGINEERING CORPORATION
```

on your screen. Amparcade is now loaded and ready to go.

3.0 SUPER LORES

We will start off our tutorial on the Amparcade language and of the Arcade Board on some very familiar ground. Most of you already know how to use the Apple's low resolution graphics mode. The Arcade Board has three graphics modes and one text mode. One of these graphics modes is very similar to Apple low resolution graphics. The Arcade Board's low resolution graphics mode differs from the Apples's lores mode in that the resolution is 64H x 48V, instead of 40H x 48V. The increased horizontal resolution makes the pixels look more like true squares, instead of rectangles as in Apple lores. There are also 16 colors available as in Apple lores mode, but not quite the same colors. Also Amparcade supports 4 different pages of Arcade Board lores graphics, (though up to 7 pages can be had), each one readily available with a single command.

Another MAJOR difference is that in Arcade Board lores 32 sprites are available. Sprites are special characters or patterns that were made specifically for super-easy animation. To move a sprite pattern, all you have to do is tell the VDP where to move the sprite to. That's all! Just tell the VDP the new horizontal and vertical positions and the sprite automatically moves there with no further effort on your part. The hardware does everything for you.

For now we will stick to more familiar things and save sprites for latter. Table 1 below shows the Apple lores commands and formats alongside the equivalent Amparcade commands. You should have no trouble in remembering the equivalent Amparcade commands as they are very similar in name, syntax and function to their Applesoft cousins. Table 2 gives the color codes for the lores command & COL.

Since the resolution of the Arcade Boards's lores screen is 64H x 48V, the legal range of values for the lores graphics Amparcade commands reflects this difference. X values may range from 0 to 63 and Y values may range form 0 to 47. The color value C can range from 0 to 15.

APPLESOFT	AMPARCADE	COMMAND FUNCTION
GR	& GM3A	SELECTS LORES GRAPHICS
	& GM3B	SELECTS LORES GRAPHICS
	& GM3C	SELECTS LORES GRAPHICS
	& GM3D	SELECTS LORES GRAPHICS
PLOT X, Y	& PLT, X, Y	PLOTS A POINT
HLIN X1, X2 AT Y	& HLN, X1, X2, Y	DRAWS HORIZONTAL LINE
VLIN Y1, Y2 AT X	& VLN, Y1, Y2, X	DRAWS VERTICAL LINE
C = SCRN (X, Y)	& SCN, X, Y, C	RETURNS SCREEN COLOR

TABLE 1 - APPLESOFT AND AMPARCADE LORES GRAPHICS COMMANDS

COLOR CODE	COLOR
0	TRANSPARENT
1	BLACK
2	MEDIUM GREEN
3	LIGHT GREEN
4	DARK BLUE
5	LIGHT BLUE
6	DARK RED
7	CYAN
8	MEDIUM RED
9	ORANGE
10	DARK YELLOW
11	LIGHT YELLOW
12	DARK GREEN
13	VIOLET
14	GREY
15	WHITE

TABE 2 - ARCADE BOARD LORES GRAPHICS COLORS

Just as a little warm up, try typing in the following little program. You should see that it works exactly as regular Apple lores would.

```
100  REM  SELECT LORES PAGE 'A', CLEAR SPRITES, BLACK BACKDROP
110  & GM3A : & SY, 0, 208 : & BCOL, 1
120  REM  SELECT RANDOM COLOR
130  & COL, RND ( 1 ) * 16
140  REM  PLOT RANDOM X - Y POINT
150  & PLT, RND ( 1 ) * 64, RND ( 1 ) * 48
160  REM  REPEAT
170  GOTO 130
```

Remember not to forget to put the ampersand sign (&) in front of each Amparcade command, and the comma before every variable or expression, including the first one.

Line 110 has three commands. The last two are probably not needed, especially if you have just powered up. They are only there to clear sprites, since we are not using any in this program, and to make the background and border color black.

When you RUN this program the screen should clear to black and start to fill up with little squares of color. This should continue until the screen is entirely filled with colored squares. You can stop the program by hitting RESET or typing Control C. If you type Control C, and have only one monitor or TV, you will not see the Apple's text screen. You can hit RESET or type:

```
& APVD
```

followed by RETURN, of course. This Amparcade command enables the Apple's video to be seen on your TV or monitor.

Try experimenting with some regular Apple lores programs, substituting the equivalent Amparcade lores commands and formats for the normal Applesoft ones. Try RUNning the program on the demo disk called 'ROD48'. This is based on an old Apple program from the old 'Red Book' called ROD'S COLOR PATTERN. You should try and compare how the colors of the regular Apple lores programs and the Amparcade programs look. The Arcade Board's colors are so much nicer and fuller looking than the regular Apple's lores colors are.

4.0 A LITTLE SOUND

Since the Programmable Sound Generator (PSG) chip is by far easier to understand than the Video Display Processor (VDP), we will play with it for a while. After we have done a few examples using the sound chip, we will return to the video chip.

4.1 SIMPLE TONES

The PSG has three channels for sound output. Each of these three channels can generate independent tones covering a 9 octave range with 16 levels of volume. The channels are labelled A, B, and C. To turn on a channel to output a tone you must do two things: specify the volume of that channel, and specify the pitch of the tone to be generated.

The volume ranges from 0 to 15 with 0 being off and 15 being the loudest volume. The pitch value ranges from 4095 to 1 with 4095 being the lowest pitch (32 Hz) and 1 being the highest (120 KHz).

Before you do anything, it is wise to initiallize the sound chip by typing:

```
&SOFF
```

This clears all tone channels and resets the volumes to 0.

Now we are going to play a tone on channel A at the maximum volume. Type in:

```
& TNA, 256, 15
```

The command & TNA, P, V stands for: play a TONE on channel A of pitch P and volume V. If you have the sound output of the Arcade Board hooked up to a monitor with an audio input or an external amplifier, you should hear a tone. You should see the Applesoft prompt '] ' and the blinking cursor, signifying that Applesoft is waiting for a keyboard input. Notice that the sound is generated automatically without any further action necessary. This is a major advantage, as once the program has set up the PSG to the desired volume and pitch, it can do other things.

Try experimenting with different values for the volume and pitch by typing in:

```
& TNA, P, V
```

where P and V are the volume and pitch respectively. Make sure V ranges only from 0 to 15 and P ranges from 1 to 4095.

You can do the same things to sound channels B and C. The corresponding commands would be:

```
& TNB, P, V
```

and

```
& TNC, P, V
```

When you want to shut off the sound type in:

```
&SOFF
```

You can even play two or three tones simultaneously using any one of the three TONE commands that you have just learned. This is done by specifying more P and V values. Try this:

```
& TNA, 256, 15, 128, 12
```

This plays two notes together. The first tone is played by channel A and has the P and V values 256 and 15 respectively. The second tone is not played by channel A, as channel A is already playing a tone. So the next channel, channel B, plays the second note, with the P and V values of 128 and 12 respectively. Here we have two tones that are exactly an octave apart. Whenever the pitch values are related to each other by a factor of 2 (divided by 2 or multiplied by 2), such as 128 and 256, then the notes are an octave apart. Why don't you try to play 3 tones simultaneously by specifying three sets of P and V values for the & TNA command, or the & TNB command or the & TNC command. It doesn't really matter which tone command you use.

4.2 MORE COMPLEX SOUNDS

Simple tones at a constant volume, or even three tones at constant volumes, are easy to create, but not very interesting. What is needed is dynamically changing tones to create more interesting and pleasing sound effects. One simple technique is a frequency sweep, where the pitch varies smoothely over an interval. Try this little program:

```
100   & SOFF : REM INITIALLIZE PSG - TURN OFF ALL SOUNDS
110   REM  SWEEP PITCH VALUE FROM 1 TO 1024
120   FOR P = 1 TO 1024
130   & TNA, P, 15
140   NEXT P
150   & SOFF : REM TURN OFF SOUND
```

Or try replacing line 150 with:

```
150 GOTO 120
```

You would have to exit this program with a Control C. With some experimentation you should be able to create 'laser blast' like sounds, or the sound of bullets whizzing by, or sirens etc. using the frequency sweep technique.

How about a bunch of random sounds. Try this program:

```
100   & SOFF : REM INIT PSG
110   & TNA, RND(1) * 1024 : REM RANDOM PITCH
120   FOR D = 0 TO 99 : NEXT D : REM DELAY LOOP
130   GOTO 110 : REM LOOP BACK
```

Or for more interesting sound sequences such as music, try this program which plays musical scales.

```
100  & SOFF : REM INIT PSG
110  RESTORE
120  READ P : REM GET NEXT PITCH FROM DATA STATEMENTS
130  IF P = 4096 THEN & SOFF : END
140  & TNA, P, 15 : REM SET PITCH VALUE CH A
150  FOR D = 0 TO 199 : NEXT D : REM DELAY LOOP
160  GOTO 120 : REM LOOP UNTIL DONE
200  DATA 428, 381, 339, 320, 285, 254, 226, 214
210   DATA214, 226, 254, 285, 320, 339, 381, 428
220  DATA 4096 : REM TERMINATOR CODE (MAX P VALUE IS 4095)
```

You can replace the pitch values in the data statements with your own values to play a song. The problem with this program is that all notes are played for the same length of time. A real song would not be like this; you would also have to specify the duration. This is easily done by placing two data elements together in the data statements. One for the pitch and the other for the duration. Use the delay loop to set the duration of the note. Why don't you try to do this as an exercise?

The volume can also be changed, as many sounds do not keep a constant volume. An organ tone is usually a constant volume as long as you hold down the key. But a piano sound dies down the instant the note is played. You can simulate these changes in volume 'manually' by varying the volume using three new commands that change volume only. These new commands are:

```
&VOLA, V
& VOLB, V
& VOLC, V
```

These command stand for VOLUME on channel A or B or C. They set the volume of the channel specified by the last letter of the command to the value V which should be between 0 to 15.

We will make use of these new commands to 'manually' change the volume of a tone as it is played. Later on we will see how this can be done automatically by using the PSG's special automatic envelope control feature. Try this:

```
100  & SOFF : REM INIT PSG
110  & TNA, 256, 15 : REM PICK ANY TONE VALUE
120  REM RAMP DOWM VOLUME SLOWLY
130  FOR V = 15 TO 0 STEP - 0.1
140  & VOLA, V
150  NEXT V
160  & SOFF
```

This creates something like a bell sound. To get a better chime-like effect, you must use a technique known as beat creation where two tones of slightly different pitch are played together.

The two tones interfere with each other in such a manner as to create an effect where the volume changes in a periodic manner. To get chimes, one tone must be played at a constant, but intermediate, volume, while the other one decays. Try this:

```
100  & SOFF
110  & TNB, 260, 6 : REM SET CH B PITCH AT LOWER VOLUME
120  & TNA, 256 : REM SET CH A PITCH SLIGHTLY DIFFERENT
130  REM RAMP DOWM CH A VOL
140  FOR V = 15 TO 0 STEP - 0.1
150  & VOLA, V
160  NEXT V
170  & SOFF
```

There are many more things that can be done. You can look at the demo disk for some ideas. Use your imagination.

NOISE GENERATION:

Often for arcade games, noise effects are more useful than tone effects. Even for music noise is useful, as drum sounds and cymbal sounds, for example, are basically noise. Gunshots and explosions are some of the sounds that can be created with noise.

There is only one noise generator on the PSG, but its output can be made to come out on any one or more of the tone channels, A, B, and C. Each of these channels can be programmed for either pure tone with no noise (use & TNA, & TNB, and & TNC only), pure noise and no tone, or a combination of both. The volume commands & VOLA, & VOLB, and & VOLC set the volume levels for both noise and tone for the given channel. The command & SOFF shuts of noise and tone on all channels as well as setting the volumes to 0 for all channels.

There are three noise commands just like there are three tone commands, one for each of the three channels. They are:

```
& NSA, NP, V
& NSB, NP, V
& NSC, NP, V
```

NP is the noise period and V is the volume (0 - 15 as usual). To set the noise, you must specify two parameters. One is the noise 'pitch' NP which varies from 1 to 31 and the other is the the volume V. The noise pitch makes the noise either richer in lower frequency components or richer in higher frequency components. Try typing this:

```
& SOFF
& NSA, 31, 15
```

You should hear a hissing sort of sound. There are three independent tone generators, so each TONE command can have its own independent pitch. But there is only one noise generator, whose output can be fed into the three sound channels. So two or more NOISE commands will not be able to make different sounds, as all that will happen is the same noise will come out of two or more channels.

One thing that really has a profound influence on the sound of the noise generator, and to a lesser degree on the tone generators as well, is the command that selects the programmable filters.

There are 16 software selectable filters numbered from 0 to 15. ALL sound output of the PSG is passed thru these filters, so when you select a given filter, all the sounds the PSG generates will be modified by this filter. Upon power up the default filter value is set to 0. All the above programs used a filter value of 0, unless you changed the filter type with the command:

```
& FILT, F
```

F is the filter number (0 to 15). There are four types of filters, each with four different frequency ranges. The four filter types are high pass, low pass, band pass, and notch pass. You do not need to know what these are to experiment with what they do to noise or tones. There are more details in the PSG Reference Manual.

Try this for noise:

```
& FILT, 2
& NSA, 1, 15
```

Try experimenting with the noise pitch and filter values with these commands:

```
& FILT, F
& NSA, NP, 15
```

where NP is some value from 1 to 31 and F is some value from 0 to 15. You could try writing a program which allows you to input the NP and F values and hear the results.

The noise can be used to make gunshots or explosions by varying the volume over time. Try this program and vary the parameters, NP, F, and D.

```
100  & SOFF : REM  INIT PSG
110  NP = 31 : REM  NOISE PITCH
120  F = 0 : REM  FILTER TYPE
130  D = 0.1 : REM  STEP SIZE FOR DELAY
140  & NSA, NP : & FILT, F : REM  SET UP NOISE PARAMETERS
150  REM  RAMP DOWN NOISE VOLUME
160  FOR V = 15 TO 0 STEP - D
170  & VOLA, V : REM  SET VOLUME OF CH A
180  NEXT V
190  & SOFF : REM  SOUND OFF
```

You might have noticed that in line 140 we didn't specify the volume of the noise. This is because the volume is optional in both the NOISE and TONE commands that we have learned so far. If you don't say what volume, then it will be whatever volume it was before.

You can also have the noise come out of channels B or C by using the commands & NSB, NP, V or & NSC, NP, V instead. It will be the same noise sound you hear, just coming out of another channel. The only advantage is that you can use one channel for tone at one volume level, and another channel for noise at a different volume level. If both noise and tone were to be at the same volume level, it would be just as easy to use the same channel for both.

You can make the sound of a bomb whistling downward and then exploding using first a frequency sweep effect followed by a noise effect where the volume is ramped down to 0 over time. Do you think you can figure out how to do this yourself? If not, there is a program on the demo disk called BOMB that you might want to look at. It does not 'manually' ramp the volume down, however, as it uses the automatic envelope control feature to do this automatically.

4.3 AUTOMATIC ENVELOPE CONTROL:

One of the interesting features of the PSG is that you can have the volume change automatically over a specified period of time by using the automatic envelope control feature. This can save programming effort as well as CPU time. In the examples before, we used 'manual' volume control to ramp the volume down (or up, if you want). The automatic envelope control feature does this automatically, and you can specify the length of time in which this is to take place, i.e. how fast it takes to ramp the volume up or down.

The envelope period is the length of time necessary for the volume to be ramped up from 0 to 15 or ramped down from 15 to 0. The shorter the period the faster this takes place. The period can range from a very short time (a few milliseconds) to about 8 seconds.

In addition there is the envelope type. There are many types of envelopes allowed. You can have the envelope ramp up from 0 volume to the maximum volume of 15 and then remain at 15 until told otherwise, or you can have just the opposite where the volume starts out at 15 and ramps down to 0 and remains there until told otherwise. You can have the volume ramp up from 0 to 15 then suddenly return to 0 without going thru the intermediate volumes, or just the opposite where the volume suddenly goes from 0 to 15 then ramps down to 0 again. You can have the volume ramp up from 0 to 15 then ramp down to 0 again, or just the opposite where the volume ramps down from 15 to 0 then ramps up to 15 again. You can also have some of these patterns automatically repeated. See the sections on automatic envelope control in the PSG Reference Manual if you want the details. For now just experiment.

There are three things that you must do to set up the automatic envelope control feature, before you can use it. First you must activate it for the channel you wish to use it on. This is done by setting the volume of that channel to 16. Volumes from 0 to 15 mean 'manual' volume control, 16 means automatic volume control (or automatic envelope control). This tells the PSG that the volume of the sound on that channel will be controlled automatically. Then the envelope period has to be specified. Last you have to tell the PSG what envelope type 0 to 15 you wish to use. Some of the numbers correspond to the same envelope type, so there are not really 16 different envelope types available.

You already know how to set the volume to 16 by using the & VOLA or & VOLB or & VOLC commands. To set the envelope period and type use the following commands:

```
& ENVP, EP
& ENVT, ET
```

EP is the envelope period value from 1 ot 65535 and ET is the envelope type value from 0 to 15. Try this program:

```
100   & SOFF : REM TURN OFF SOUND
110   & VOLA, 16 : REM MAKE CH A VOL UNDER AUTOMATIC ENVELOPE CONTROL
120   & TNA, 256 : REM SET UP CH A FOR TONE AND SELECT PITCH
130   P = 4096 : REM ENVELOPE PERIOD
140   T = 0 : REM ENVELOPE TYPE
150   & ENVP, P : REM SET ENVELOPE PERIOD
160   & ENVT, T : REM SET ENVELOPE TYPE
```

NOTE: Do not end this program with & SOFF, or the sound will die before it is finished. Try experimenting with different values of P (0 to 65535) and T (0 to 15). Also you can replace line 120 with a & NSA command to see the effects on noise instead of on tone. Try using this feature to simulate a bomb explosion.

The command & ENVT not only selects the envelope type, but it also triggers the PSG to create a tone or noise with the specified envelope type every time it is executed. For instance an envelope type of 0 produces a sound that decays form maximum volume to 0. Once the sound has decayed to 0 volume, you can not hear it. In order to repeat the sound you would send the envelope type back to the PSG with the & ENVT command. This will trigger the PSG to make the sound again.

4.4 FILTERS

The filters are used to modify the sound output. The PSG upon power up has the filter value 0. This corresponds to the lowest cut off frequency high pass filter. The high pass filter values, 0, 4, 8, and 12 emphasize the high frequencies and give the tone more treble. The low pass filter values 1, 5, 9, and 13 emphasize the low frequencies and give the tone more bass. The band pass filters values, 2, 6, 10, and 14 make the tone sound horny, while the notch pass filters 3, 7, 11, and 15 emphasize both the high and low frequencies while attenuating the in between frequencies. The command to select a filter type is:

```
& FILT, F
```

Where F is a number from 0 to 15. The filter type affects ALL the sounds made by the PSG. There is only one filter for all the channels.

5.0 MORE GRAPHICS

There are three graphics modes of the VDP. We already played around with the multicolor graphics mode in Section 3.0 Super Lores. Now it's time to move on to graphics mode 1. After that we'll get to sprites. Graphics mode 1 is a little bit more complicated than the lores mode, so we will explain it first. It may seem odd at first, but once you get the hang of it, it's quite simple and flexible.

In Graphics Mode 1 the screen is divided into 32 horizontal blocks and 24 vertical blocks. These blocks are called 'tile' positions or charcater cells. There are a total of 32 x 24 = 768 different tile positions on the screen. Each one of these tile positions is an 8 x 8 pixel block. That is, within each tile position there are 8 horizontal pixels by 8 vertical pixels for a total of 64 pixels per tile position. Each of these pixels can be either on of off.

You can think of Graphics Mode 1 as being similar to the Apple's text page. In the Apple's text page, the screen is divided into 40 horizontal (for 40 character across) by 24 vertical (for 24 lines of 40 characters) character cells. That is a total of 40 x 24 = 960 character cells. These character cells are further subdivided into 7 horizontal by 8 vertical pixels. This is why the Apple's hires graphic's black and white resolution is 280H (40 x 7) by 192V (24 x 8). Into each of these 7 x 8 pixel character cells you can place any one of the available Apple text characters, which are composed of a matrix of 6 x 7 pixels, as space between adjacent characters and lines is provided. These characters are either normal ASCII characters (less lower case characters), inverse video characters, or flashing characters. You have no other choices. Also the characters must be black and white only, as no colored text is available.

Other computers offers some 'graphics characters' along with the standard ASCII characters. The TRS-80, Atari and Commodore computers are like this. This is how graphics is done on these computers, with 'characters'. It's like doing graphics with asterisks, periods, commas, alphabet characters, etc. except you also have special 'characters' for graphics such as squares, crosses, angles, vertical line segments, horizontal line segments, various right angle corner pieces, etc. All the characters available in the computer make up what is called 'the character set'. The Apple character set has no lower case or special graphics characters. The information for the character set is stored in the character generator ROM. Since this is a ROM, the data is permanent, it can not be changed.

Graphics Mode 1 is similar to this as it is a type of character graphics. There are some major differences though that make Graphics Mode 1 far more powerful than simple character graphics. Most importantly is the fact that the 'character set' is not stored permanently and unchangeably in ROM, but is user-defined and stored in RAM, and thus can be altered. This allows for a great deal of flexibility as you can design your own 'character sets'. These character sets can be anything you want, as long as they are within an 8 x 8 pixel block. They need not have anything to do with normal ASCII text characters. Another advantage is that you can specify the colors of these character sets. Each character has a foreground color and a background color which can be selected by the user. Another advantage is that several 'character sets' can reside in memory at one time, and you can choose between them.

At any given time you can have 256 different characters patterns defined in the currently active 'character set'. These characters patterns are called 'tile patterns' or just 'tiles'. There is a table in the video RAM on the Arcade Board called the Pattern Generator Table, which defines the patterns of the 256 different tiles available. Eight bytes are needed for each tile, so a total of 256 x 8 = 2048 (2K) bytes are needed to completely specify the Pattern Generator Table. The patterns are numbered from 0 to 255. If you want to have regular text patterns, then you would make a given tile pattern have the appropritate shape for the ASCII character of the same number. For example a capital 'A' has the ASCII value 65. You would make tile pattern 65 have the shape for a 'A'. However, you can define any patterns you want, and if you have no need of text, then the patterns can be more appropriate for your uses.

The colors of the tile patterns are specified in a table called the Color Table. There are only 32 different elements in this table, so there can be only 32 different combinations of foreground / background color tile patterns. Only two colors are allowed within a tile pattern. The Pattern Generator Table is organized such that patterns 0 - 7 ALL have the same foreground / background color combinations specified by Color Table entry 0. Tile patterns 8 - 15 have the same foreground / background combination specified by Color Table entry 1 and so forth. Each group of eight patterns, starting from pattern # 0, have the same color combination specified by the Color Table entry # equal to the integer portion of the pattern # divided by 8.

If the above is too confusing, you should read the description on Graphics Mode 1 in the VDP Reference Manual.

However, you may make an analogy of Graphics Mode 1 by pretending that you are a tile setter doing a kitchen counter top design. The counter top is a grid of 32 horizontal by 24 vertical squares. Into this grid you must make a design by placing ceramic tiles into the squares. You devise a coordinate system for each of the squares starting with the origin in the upper left hand corner. The X values range from 0 to 31 (left to right), while the Y values range from 0 to 23 (top to bottom), to correspond with the 32 horizontal by 24 vertical tile positions. See Figure 4 in the VDP Reference Manual Section 3.3.

You are limited in your choice of tiles as you only have 256 different tile patterns to chose from, and each tile may be only two colors from a choice of 16 different colors. Also these tiles are restricted in that they have a definite top and bottom, that is you can not take a certain tile and place it upside down in one location, right side up in another, side ways at another and so forth. They must all be placed right side up.

In addition the tiles are placed in bins in front of you. There are 32 bins, each with eight different tile patterns in them. The tiles in each bin are all of the same two colors. You may number these bins from left to right as bin 0, bin 1, ... up to bin 31. The tiles in each bin may be numbered 0 to 7 for bin 0, 8 to 15 for bin 1, 16 to 23 for bin 2, and so forth, up to 248 to 255 for the tiles in bin 31.

As you have an apprentice tile setter, you do not need to dirty your hands with the job, so you tell your apprentice what to do. You say "at location (0, 0) place tile pattern # 69, at location (1, 0) place tile pattern # 45, at locations (2, 0), (3, 0) and (4, 0) place tile pattern # 56, ..., and at location (31, 23) place tile pattern # 238." In this way you tell your apprentice how to lay out the tile pattern design that you desire to create.

This is what Graphics Mode 1 is like. There are 32 color sets (bins) consisting of 8 tile patterns each. All the tiles in a color set have the same two colors. One color for the foreground and one color for the background. You must first specify the tile patterns for each color set, and the two colors for that color set. If you change the color of a color set, ALL the patterns in that color set will have the new colors. You then specify for each of the 768 different (X, Y) locations what tile pattern # goes there.

The command to define a tile pattern is:

```
& DTIL, P, A, B, C, D, E, F, G, H
```

Where P is the tile pattern number 0 to 255 and the eight values A thru H represent the pattern data. The pattern data bytes each represent a row of the 8 x 8 pixel pattern. Byte A is the top row of 8 pixels, and byte H represents the bottom row of 8 pixels. All the bits that are 1's are set equal in color to the 'ones' color defined in the appropriate color set in the Color Table. All the bits that are 0's are set equal to the 'zeros' color. The 'zeros' color is the background color, while the 'ones' color is the foreground color. See the VDP Reference Manual, Section 9.0, for some examples on creating tile patterns.

The command that sets the colors of the color set is:

```
& CSET, CS, C
```

Where CS is the color set # 0 to 31 and C is the color 0 to 255. The 4 most significant bits of C represent the 'ones' color and the 4 least significant bits of C represent the 'zeros' color. If you do not know binary or hexadecimal, then it is recommended that you learn it as there is no way to understand bit patterns without understanding how to count in binary or hex. For now, though you may use this formula for setting the foreground and background colors:

```
C = FC * 16 + BC
```

Where FC is the foreground color value and BC is the background color.

Once you have defined as many tile patterns and color sets as you need to use, then you would use this command to set the particular tile pattern at a given (X, Y) location:

```
& STIL, X, Y, P
```

The X and Y values are the coordinate positions of the location (X, Y) and P is the pattern number 0 to 255.

One of the first things you have to do before you can see any graphics is to set up the VDP registers. This process tells the VDP what graphics mode to display, the locations of the various tables in the 16K of video RAM (VRAM) necessary to define the graphics images, the backdrop color, the sprite size and magnification, etc.

This can easily be done with one command:

```
100  & GM1A
```

This command sets up the VDP to Graphics Mode 1, page 'A' and places the five tables in VRAM at specific locations. There are four different pages (A, B, C and D) of graphics mode 1 displays.

5.1 CREATING TILE PATTERNS

To create tile patterns you must have an understanding of binary numbers and how to convert them to decimal numbers. Texas Instruments and Hewlett-Packard make special calculators for this purpose, but it is easily done on paper or in your head, once you get used to it. Section 9.0 of the VDP Reference Manual describes a step-by-step approach to creating tile patterns. You may want to look this over at this time. The binary numbers used to create the tile patterns in this section are converted to hexadecimal numbers. For the Amparcade commands you will need to convert them to decimal numbers.

The tile patterns are made up of 8 rows of 8 pixels each for a total of 64 pixels per tile pattern. To define a tile pattern, you need to specify 8 numbers in the range 0 to 255 (since 255 is the highest number you can get from 8 bits), each number represents a row of the tile pattern. For instance a box pattern like this:

TILE PATTERN								DECIMAL NUMBER
1	1	1	1	1	1	1	1	255
1	0	0	0	0	0	0	1	129
1	0	0	0	0	0	0	1	129
1	0	0	0	0	0	0	1	129
1	0	0	0	0	0	0	1	129
1	0	0	0	0	0	0	1	129
1	0	0	0	0	0	0	1	129
1	1	1	1	1	1	1	1	255

FIGURE 1 - TILE PATTERN CREATION

is specified by the numbers 255, 129, 129, 129, 129, 129, 129, 255 in decimal.

Once you have the decimal numbers that make up the tile pattern, you can define the tile pattern with the & DTIL command. The next step is to define the color of the tile pattern. You have to know what color set the tile pattern belongs to. Remember a color set consists of 8 tile patterns, all with the same '0' and '1' colors. To get the color set number CS for any tile pattern P, just divide P by 8 and take the integer portion only.

CS = INT (P / 8)

Now you can define the '0' and '1' colors of the 8 tile patterns in the color set CS using the & CSET command. Once the pattern and colors are defined you are ready to place the tile into any one of the 768 different character cells on the screen.

Amparcade has several different commands to do this kind of thing. Some commands allow you to place a single tile in a single location, others allow you to create horizontal rows or vertical columns of the same tile pattern, others allow you to do the same with different tile patterns, and one command allows you to fill the entire screen with the same tile pattern. The basic command, though, is the & STIL command. It allows you to place a singe tile pattern P at any X - Y location on the screen:

```
& STIL, X, Y, P
```

This short program defines one tile pattern, pattern 0, and fills up the entire screen with it:

```
100  & GM1A : & SY, 0, 208 : REM  SET UP GR MODE 1
110   & DTIL, 0, 255, 129, 129, 129, 129, 129, 129, 255 : REM
     DEFINE TILE PATTERN 0
120  & CSET, 0, 106 : REM DEF 0 AND 1 COLORS
130  REM   FILL SCREEN WITH TILE PATTERN 0
140  FOR X = 0 TO 31 : FOR Y = 0 TO 23
150  & STIL, X, Y, 0 : REM SET TILE PAT 0 AT (X, Y)
160  NEXT Y : NEXT X
```

When you RUN this program, you should see the whole screen fill up with a box like pattern. The colors were defined so that the sides of each box pattern are red and the insides are yellow. You can try replacing the 106 in line 120 with other values from 0 to 255 and see the colors change.· Or try different patterns. This program is basically how you go about creating a picture in Graphics Mode 1. You define your patterns, you define the colors, and then you place the various tile patterns in the 768 different tile positions. To create more complex graphics you would use more than one pattern and different colors too.

Again we have made use of the command & SY, 0, 208 without explaining it to you. This is a sprite command and is necessary to deactivatie the sprites, since we are not using any in this program. If we neglect to do this you may see as many as 32 'garbage' sprites on the screen. We will explain this in the next section. If you only have one monitor or TV, to get back to see regular Apple video you can hit reset or type (blindly) & APVD, followed by a carriage return. Apple video should again appear on the screen.

We could have filled the screen using the & FILL command. This would have been a lot faster and easier. You might want to replace lines 140, 150 and 160 of the above program with the line:

```
140  & FILL, 0
```

This fills up the entire screen with pattern 0. You might not notice it working unless you change the color of the color set, defined in line 120. Otherwise it will draw the same picture over one that already exists and you won't even notice a change. Maybe you can change line 120 to:

```
120  & CSET, 0, RND ( 1 ) * 256
```

and add line 150:

```
150  GOTO 120
```

Now when you run this program the screen should rapidly fill with different colored boxes until you hit RESET or Control C.

The demo disk has more programs that can be studied as examples as to how to use the & DTIL, & CSET, & STIL and other commands. See, for example, COLOR BARS, TILES, BRICKS, SQR and PATCHES.

5.2 SPRITES AND ANIMATION

One of the most powerful features of the Arcade Board is the ease with which you can animate objects. The VDP has the capability to move up to 32 different objects, called sprites, merely by specifying the X and Y coordinates of the object. As soon as you specify the X and Y coordinates of a sprite, the sprite automatically appears at that location on the screen without any additional effort on your part. For sprites the X and Y coordinates range from 0 to 255 for X and 0 to 191 for Y. In addition there are ways to extend this range slightly, see the VDP Reference Manual, Section 3.7, where sprites are explained.

Sprites come in two sizes and two magnifications. Size 0 sprites are 8 x 8 pixel patterns. With magnification set to 0 they appear as 8 x 8 pixel sprites, but when set to 1 they are doubled in size to 16 x 16 pixel sprites. The resolution, however, remains the same 8 x 8. Size 1 sprites are 16 x 16 pixel patterns. When magnification is set to 0 they appear as 16 x 16 pixel sprites, but when set to 1 they are doubled in size to 32 x 32 pixels. The resolution is still 16 x 16.

The size and magnification are set with the Amparcade command & MAG. Even thought it is called 'MAG' is sets both size and magnification. The format is:

```
& MAG, M
```

M	SIZE	MAGNIFICATION	SPRITE
0	0	0	8 x 8
1	0	1	16 x 16
2	1	0	16 x 16
3	1	1	32 x 32

TABLE 3 - SPRITE SIZE AND MAGNIFICATION

The first thing to do in using sprites is to define their patterns. There can be up to 256 size 0 or 64 size 1 sprite patterns available at any given time. For now we will stick to size 0 sprites. Defining size 0 sprite patterns is identical to defining tile patterns. Both tile patterns and size 0 sprites are 8 x 8 pixel patterns. In fact, sprites are essentially movable tile patterns. Like tile patterns you must first specify the patterns. There are 256 possible sprite patterns, just as there are 256 possible tile patterns. In fact, you can use the same patterns for tiles as for sprites.

You can have up to 256 different size 0 sprite PATTERNS, but at any given time you can have, at most, 32 SPRITES. A sprite and a sprite pattern are not the same thing. Any of the 32 sprites can assume any one of the 256 available sprite patterns. In fact you can use the same sprite pattern for all 32 sprites.These 32 sprites are numbered from 0 to 31. You must specify the X an Y positions of each sprite that you are using, the color of that sprite, and which of the 256 sprite patterns will be 'carried' by that sprite. Any sprite, say sprite 10, can have any one of the 256 different patterns. The sprite numbers 0 to 31 and the sprite pattern numbers 0 to 255 are two different things. Sprite 0 can have any one of the 256 different sprite patterns, sprite 1 can also have any one of the 256 different sprite patterns, etc.

You can think of the graphics image generated by the VDP as a series of 35 planes of glass, which you are looking through head on. The 32 planes nearest you are reserved for sprites only. Refer to Figures 2 and 3 in Section 3.1 of the VDP Reference Manual. You may also want to read the section, as well as section 3.7, for more details of the sprite planes and sprites in general. The 33 rd plane is the pattern plane onto which you draw background patterns in graphics mode 1 or 2 or multicolor mode. The 32 sprite planes each may have only one sprite on it. These planes are transparent, except where the sprites are.

The sprites with lower numbers have higher priority than those with higher numbers. Sprite 0 has the highest priority and resides on the first sprite plane (plane 0). Sprite 31 has the lowest priority and resides on the last sprite plane (plane 31). Priority is important in cases when two or more patterns are overlapping each other at the same screen locations. The pixels of the higher priority sprite will be seen, while the overlapping pixels of the lower priority sprite will temporarily be obscured. This is useful for a pseudo 3 D effect.

This example program shows how to use sprites. It defines a sprite and moves it across the screen. First of all you must define the background plane over which the sprites move. This is the pattern plane. The program above filled the pattern plane with box tile patterns. We will assume that you have already done this. If so add the following code to the above program:

```
200  & MAG, 1 : REM  SET SIZE 0 MAG 1 SPRITE
210   & DSPR, 0, 255, 129, 129, 129, 129, 129, 129, 255 : REM
     DEFINE SPRITE PATTERN 0 AS A BOX SHAPE
220  & SY, 1, 208 : REM DISABLE SPRITES 1 AND BEYOND
230  REM  MOVE SPRITE 0 ACROSS SCREEN AT Y = 80
240  FOR X = 0 TO 255
250  & SSPR, 0, 80, X, 0, 3
```

```
260  NEXT X
270  GOTO 240 : REM REPEAT
```

When you RUN this program, you should see a green box like sprite moving smoothely across the screen at Y position 80. The background pattern plane should be filled with boxes from the box program above. However, you may fill the pattern plane with any pattern, before you run this program.

Line 210 defines the sprite pattern. & DSPR is exactly like the & DTIL command. It defines the sprite pattern specified, to the eight following bytes of data. Here sprite pattern 0 is defined to be a box. We do not need to specify any more sprite patterns, since we are only using one sprite pattern.

Line 250 in the FOR NEXT loop is the line which actually moves the sprite. This command & SSPR (for set sprite) sets the sprite number specified to the Y, X position specified, and tells it what shape to have and the color. The format for this command is:

```
& SSPR, S, Y, X, P, C
```

where S is the sprite number (here 0), Y and X are the vertical and horizontal positions, P is the sprite pattern number (here also 0, but could be any sprite pattern number from 0 to 255), and C is the color (here 3 for green). Only the X value varies in this FOR NEXT loop, so the sprite moves horizontally across the screen.

Line 220 is necessary whenever you are not using all 32 sprites. This line terminates the sprite processing of the VDP. Without it the VDP has no way of knowing that you do not want to use all 32 sprites. It would put all 32 sprites on the screen as specified by whatever happens to be in memory in the sprite attribute table (see Section 3.7 of the VDP Reference Manual) at the time. If this is random garbage, then you will see random garbage on the screen. The method to terminate sprites is to place a 208 in the Y position byte of the first sprite not used. We are only using sprite 0 in this demo, so the first sprite not used is sprite 1. We set the Y position of sprite 1 to 208, and sprite 1 and all sprites following (sprites 2 to 31) are not used.

There are many sprite commands in Amparcade. & SSPR is the most general and allows you to specify all four parameters of a sprite, its X and Y positions, its pattern, and its color. There are commands that specify only one parameter, such as & SY for the Y position, & SX for the X position, & SP for the pattern, and & SC for the color.

There are many examples of sprite usage in the demo programs. See SCRAMBLE and UFO for instance.

ARCADE BOARD AMPARCADE

COMMAND DOCUMENTATION

1.0 NOTATION USED IN COMMAND FORMATS

1.1 LEGAL VARIABLE TYPES AND VALUES

All expressions following Amparcade commands must be real variables or numeric constants. No integer or string variables are allowed. In order to increase execution speed, there is minimal error checking, therefore if the wrong variable types are used, no error messages will be generated. In most commands only expressions whose values range from 0 to 255 are allowed. Values out of range will result in an 'ILLEGAL QUANTITY ERROR' message. Some commands allow 16 bit values from 0 to 65535.

All variables must be preceded by a comma, even if it is the first variable following a command. Omission of this comma will result in a 'SYNTAX ERROR IN LINE xxx' message.

1.2 OPTIONAL QUANTITIES

If a variable is surrounded by square brackets, such as [, A], then specification of the variable is optional. All other variables in the command must be specified, or else a 'SYNTAX ERROR' message will result.

A sequence of variables, each individually surrounded by square brackets, such as [, A] [, B] [, C], means that specification of each variable is optional, but that all preceding variables in the sequence must first be specified. For example, the command:

```
& TNA, P1,  [, V1 ]  [, P2 ]  [, V2 ]  [, P3 ]  [, V3 ]
```

requires that variable P1 is always specified, but the five other variables are optional, and need not be specified. However, if specification of variable V2, for example, is desired, then all the other optional variables preceding V2 (V1 and P2) must first be specified. Variabes P3 and V3 need not be specified, since they do not precede V2.

In some commands the choice is between specifying ALL of the optional variables as a complete set, or specifying NONE of them. It is all or nothing, no in betweens. This is indicated by the symbol: [, ...] . This means that all the required variable specifications of the command may be repeated, only as a complete set. For example the command:

```
& STIL,X, Y, P  [, ... ]
```

requires that the three variables X, Y, and P be specified. However, one or more complete sets of variables may also be specified, if desired, such as:

```
& STIL, X, Y, P, X2, Y2, P2
```

2.0 COMMANDS AFFECTING VDP REGISTERS

& APVD

APPLE VIDEO: This command is for users of one monitor or TV only. It enables the Apple's video signal to pass through the Arcade Board unchanged and on to the monitor or TV.

& MAG, M

MAGNIFICATION: This command sets the sprite size and magnification. M varies from 0 to 3. M = 0 sets sprites to size 0 and magnification 0, that is 8 x 8 pixel sprites. M = 1 is size 0 and magnification 1, that is 8 x 8 pixel sprites, but double sized pixels. M = 2 is for size 1 and magnification 0 sprites, that is 16 x 16 pixel sprites. M = 3 is for size 1 and magnification 1 sprites, that is 16 x 16 pixel sprites, but double sized pixels.

& BCOL, C

BACKDROP COLOR: This command changes the backdrop color to the value given by the low 4 bits of the 8 bit value C, and also determines the text color, which is given by the high 4 bits of C. It writes the value C into VDP register 7, which control the text / backdrop color. C ranges from 0 to 255, but values in the range 0 to 15 are needed to change the backdrop color to one of the possible 16 colors. See the VDP Reference Manual, Section 3.0, for a list of the colors and their values.

& RSTA, S

READ STATUS: This command reads the VDP status register and returns its value in the variable S. This register is used to check for the frame flag, the collision flag, the sprite coincidence flag, and the fifth sprite number. Refer to the VDP Reference Manual, Section 2.2, for an explanation of these flags.

& GM1A
& GM1B
& GM1C
& GM1D

GRAPHICS MODE 1: These four commands set up the VDP registers for graphics mode 1 and allocate video RAM (VRAM) to the name table, color table, pattern generator table, sprite attribute table, and sprite pattern generator table. The backdrop color, sprite size and sprite magnification remain unchanged. The screen is not cleared by any of these commands and none of areas of VRAM occupied by the various tables are changed from their initial values.

The VRAM may be allocated in such a way as to provide four totally independent pages (A, B, C and D) of graphics mode 1 screens. Each of these pages has its own name table, color table, pattern generator table, sprite attribute table and sprite pattern generator table. There is a small price to pay for the total independancy of each of the four pages of graphics mode 1. The sprite pattern generator table is overlapped by the name table, the sprite attribute table, and the color table. Therefore there is room for only 136 different sprite patterns. These are sprite patterns 0 to 127 and 148 to 255. The area normally used by sprite patterns 128 to 147 is now used for the name table, the sprite attribute table, and the color table.

Each page occupies a 4 K block of VRAM. Page A uses up the first 4 K block, page B uses the second 4 K block, page C uses the third 4 K block, and page D uses the last 4 K block. See the VDP Reference Manual, Section 5.1, for a memory map of how the video RAM is allocated for each of these four pages of graphics mode 1.

& GM2

GRAPHICS MODE 2: This command sets up the VDP to the only page of graphics mode 2 available. There is room for 256 different sprite patterns. See the VDP Reference Manual, Section 5.2, to see how the various tables are allocated by this command.

& GM3A
& GM3B
& GM3C
& GM3D

GRAPHICS MODE 3: These commands are similar to the 'GR' command of regular Apple low resolution graphics, except there are four totally independent pages (A to D) of multicolor mode graphics. Each command sets up the VDP registers for multicolor mode graphics and allocates video RAM (VRAM) to the name table, pattern generator table, sprite attribute table, and sprite pattern generator table. The color table is not used in this mode. The screen is automatically cleared to transparent by filling the pattern generator table with 0's. The name table is set up in the suggested manner for multicolor mode graphics. See the VDP Reference Manual, Section 3.5, for more details on this. The backdrop color, sprite size and magnification are unchanged.

There is room in the 16 K of VRAM for four totally independent pages of multicolor mode graphics. Each page includes its own name table, pattern generator table, sprite attribute table, and sprite pattern generator table. There is a small price to pay for the total independancy of each of the pages of multicolor mode graphics. Each page has its own sprite pattern generator table, but there is room for only 160 of the possible 256 sprite patterns in this table. Sprite patterns 0 to 127 and 224 to 255 can be used. Sprite patterns 128 to 223 can not be used, as the area normally used to hold these patterns is used by the name table. See the VDP Reference Manual, Section 5.3, for more informatation.

Each page occupies a 4 K block of VRAM. Page A occupies the first 4 K block, page B occupies the second 4 K block, page C occupies the third 4 K block, and page D occupies the last 4 K block. There is some overlapping of tables in order to cram all the relevant tables into 4 K of VRAM. This is why there is room for only 160 different sprite patterns - the name table overlays part of the sprite pattern generator table. See the VDP Technical Reference section, under VRAM Allocation, for a memory map of how the video RAM is allocated to the various tables by these commands.

& VREG, REG #, D [, D]

VDP REGISTERS: This command is used to load one or more VDP registers with the value(s) D. REG # is the VDP register number (0 - 7) and D is the data to be written to that register. Any further optional data values D will be written to the next higher numbered VDP register. Be careful not to specify more than 8 - REG # data values, otherwise an error will result. If REG # is 0, then at most eight data values may be specified. If REG # is 7, then only one data value can be specified.

3.0 COMMANDS AFFECTING THE NAME TABLE

& STIL, X, Y, P [, ...]

SET TILE: This command is used to set tile pattern P at the character cell location (X, Y). It writes the value P into the name table at the appropriate location corresponding to the screen position (X, Y). This command is used in both graphics mode 1 and graphics mode 2. X ranges from 0 to 31 and Y ranges from 0 to 23. P ranges from 0 to 255 for graphics mode 1 and from 0 to 767 for graphics mode 2.

There are restrictions on the placement of tile patterns in graphics mode 2. Tile patterns 0 to 255 can be placed only in the top third of the screen (Y =0 to Y = 7), tile patterns 256 to 511 can be placed only in the middle third of the screen (Y = 8 to Y = 15), and tile patterns 512 to 767 can be placed only in the bottom third of the screen (Y = 16 to Y = 23). See the VDP Reference Manual, Section 3.4, for more details.

& GTIL, X, Y, P [, ...]

GET TILE: This command is used to read the number of the tile pattern which is currently at character cell location (X, Y). It reads the name table location which corresponds to the screen position (X, Y), and returns the tile pattern number in the variable P. This command is used in both graphics mode 1 and graphics mode 2. X ,Y, and P have the same meanings and ranges as in the command: & STIL.

& HROW, X, Y, P, N

HORIZONTAL ROW: This command is used to create a horizontal row of identical tile patterns, specified by P, starting at screen position (X, Y). It writes the value P into sequential addresses of the name table N times starting at the address corresponding to the screen position (X, Y). The variable N is the number of tile patterns P to be placed in the row. If N + X is greater than 32 then the patterns will appear on the next row down. If N = 1 then this command becomes identical to the & STIL command. If N = 0 then 256 tile patterns P will be drawn in sequential rows. N ranges from 0 to 255. X, Y, and P have the same meanings and ranges as in the command: & STIL. This command is used in both graphics mode 1 and graphics mode 2.

The restrictions on the placement of tile patterns in graphics mode 2 , mentioned under the & STIL command definition, apply here as well. In graphics mode 1 & HROW will always create a horizontal row, or several rows, of IDENTICAL tile patterns. This will not always be the case, however, in graphics mode 2. When multiple horizontal rows are drawn and a 'one-third screen boundary' is crossed, tiles plotted across the boundary will have patterns defined by P + 256 rather than P. The tile pattern corresponding to this new number will show up on the screen. For example this command:

```
& HROW, 0, 7, 0, 64
```

will draw two full rows of 32 tile patterns each starting at screen position (0, 7). The first row will extend from (0, 7) to (31, 7), and the second row will extend from (0, 8) to (31, 8). As you would expect, the top row will consist of tile pattern 0, but since a 'one-third screen boundary' is being crossed, the second row will consist of tile pattern 0 + 256 = 256. This problem also has to be taken into consideration when drawing vertical columns, using the & VCOL command. See the VDP Reference Manual, Section 3.4, for more details.

& VCOL, X, Y, P, N

VERTICAL COLUMN: This command is used to create a vertical column of identical tile patterns, specified by P, starting at the screen position (X, Y). It writes the value P into the N addresses of the name table, starting at the address corresponding to the screen postion (X, Y). The next address to write the value P into is calculated by adding 32 to the last name table address written to. The variable N is the number of tile patterns P to placed in the column. N + Y should not be greater than 24 or other VRAM tables may be inadvertantly written into. If N = 1 then this command becomes identical to the & STIL command. N has a maximum range from 1 to 24 ONLY, and Y + N should never be greater than 24. The meaning and ranges of X, Y, and P are the same as in the & STIL command. This command is used both in graphics mode 1 and graphics mode 2. The same restrictions for graphics mode 2, mentioned in the & HROW command, apply here as well.

& HTIL, X, Y, PO .[, Pl] [, P2] [, P3] etc.

HORIZONTAL TILES: This command is used to place a horizontal row of different tile patterns, specified by the variables PO, Pl, etc. starting at tile position (X, Y). It writes the different values PO, Pl, etc. into sequential addresses of the name table, starting at the address corresponding to the screen position (X, Y). The meanings and ranges of X and Y are the same as in the & STIL command. This command is used in both graphics mode 1 and graphics mode 2. The various P's range from 0 to 255 for graphics mode 1, and from 0 to 767 for graphics mode 2. The restrictions in graphics mode 2 usage, mentioned in the & HROW command, also apply here.

& VTIL, X, Y, PO [, Pl] [, P2] [, P3] etc.

VERTICAL TILES: This command is used to place a vertical column of different tile patterns, specified by the variables PO, Pl, P2, etc. starting at tile position (X, Y). It writes the values of PO, Pl, P2, etc. into the name table starting at the location corresponding to screen position (X, Y). Subsequent addresses are calculated by adding 32 to the last address written to. This command is used in both graphics mode 1 and graphics mode 2. The meanings and ranges of X, Y, and various P's are the same as in the & HTIL command. The restrictions in graphics mode 2, mentioned in the & HROW command, apply to this command as well.

& FILL, P

FILL: This command is used to fill the entire screen with the pattern P. All 767 tile positions of the screen will be identical. This command is used to normally clear the screen to some pattern P which is defined to be all black, but can be used to fill the screen with any pattern P. This command is used for graphics modes 1 and 2 only. P ranges from 0 to 255 for graphics mode 1 and from 0 to 767 for graphics mode 2.

4.0 COMMANDS AFFECTING THE COLOR TABLE

& CSET, CS, C [, ...]

COLOR SET: This command is used only in graphics mode 1 to define the '0' and '1' colors of the 32 unique color sets which in turn define the colors of the 256 different tile patterns used in graphics mode 1. The value C is written into the graphics mode 1 color table at the appropriate location for color set CS. See the VDP Reference Manual, Section 3.3, for more details on color sets and how they determine the colors of the tile patterns. CS ranges from 0 to 31 and C ranges from 0 to 255. The low four bits of C determine the '0' color and the high four bits determine the '1' color.

& RCST, CS, C [, ...]

READ COLOR SET: This command is used only in graphics mode 1 to obtain the '0' and '1' colors of color set CS. It reads the data from the graphics mode 1 color table, and returns the value in the variable C. The meanings and ranges of CS and C are identical to those in the command: & CSET.

& TCOL, P, C0, C1, C2, C3, C4, C5, C6, C7 [, ...]

TILE COLOR: This command is used to define the '0' and '1' colors of each of the eight rows of tile pattern P. It writes the eight values C0 to C7 into the graphics mode 2 color table at the appropriate locations for tile pattern P. C0 defines the '0' and '1' colors for the top row (row 0) of tile pattern P, C1 defines the '0' and '1' colors for the second row (row 1), C2 defines the

'0' and '1' colors for the third row (row 2), etc. This command is used only in Graphics Mode 2, where each of the 768 different tile patterns may have its own unique set of colors, independent of the colors of the other 767 tile patterns. See the VDP Reference Manual, Section 3.4, for more information on how the tile patterns are colored. P ranges from 0 to 767 and each of the variables C0 to C7 range from 0 to 255. The low four bits of each of the variables C0 to C7 define the '0' color for the row, and the high four bits define the '1' color for the row.

```
& RTCL, P, C0, C1, C2, C3, C4, C5, C6, C7  [, ... ]
```

READ TILE COLOR: This command is used only in graphics mode 2 to obtain the '0' and '1' colors of each of the eight rows of tile pattern P. It reads the eight bytes of data, which define the '0' and '1' colors of each of the eight rows of tile pattern P, from the graphics mode 2 color table, and returns these eight bytes of data in the variables C0 to C7. The meanings and ranges of P and C0 to C7 are the same as in the command: & TCOL.

5.0 COMMANDS AFFECTING THE PATTERN GENERATOR TABLE

```
& DTIL, P, P0, P1, P2, P3, P4, P5, P6, P7  [, ... ]
```

DEFINE TILE: This command is used to define the shape of tile pattern P according to the values of the eight variables P0 to P7. P0 defines the top row (row 0) of tile pattern P, P1 defines the second row (row 1), P2 defines the third row (row 2), etc. These 8 bytes of data are written into the pattern generator table at the appropriate locations for tile pattern P. P ranges from 0 to 255 for graphics mode 1, and from 0 to 767 for graphics mode 2. P0 to P7 range from 0 to 255. For example this command:

```
& DTIL, 0, 255, 129, 129, 129, 129, 129, 129, 255
```

defines tile pattern 0 as a square box of 8 pixels per side. See the VDP Reference Manual, Sections 3.3, 3.4 and 9.0, for more details on defining tile patterns.

```
& RTIL, P, P0, P1, P2, P3, P4, P5, P6, P7  [, ... ]
```

READ TILE: This command is used to read the eight bytes of data which define each of the eight rows of tile pattern P. These eight bytes are read from the pattern generator table, and returned in the eight variables P0 thru P7. All parameters have the same meanings and ranges as in the & DTIL command.

& COL, C

COLOR: This command is similar to the Apple low resolution graphics command: COLOR = C. It sets the color of subsequent points or lines to be plotted to the color corresponding to the value C. C ranges from 0 to 15. It is used only in multicolor graphics mode. This command does not directily affect the pattern generator table, but does so indirectly through the & PLT, & HLN, and & VLN commands.

& PLT, X, Y

PLOT: This command is similar to the Apple low resolution graphics command: PLOT X, Y. It plots a point of the last selected color (defined by the & COL command) at the point defined by X and Y. X ranges from 0 to 63 and Y ranges from 0 to 47. It is used only in multicolor graphics mode.

& SCN, X, Y, C

SCREEN: This command is similar to the Apple low resolution graphics command: SCRN (X, Y). It returns the color value of the point whose screen coordinates are (X, Y) into the variable C. The ranges of X and Y are the same as in the & PLT command. It is used only in multicolor graphics mode.

& HLN, X1, X2, Y

HORIZONTAL LINE: This command is similar to the Apple low resolution graphics command: HLIN X1, X2 AT Y. It draws a horizontal line in the last selected color from the point (X1, Y) to the point (X2, Y). X1 and X2 range from 0 to 63, but X2 must be greater than X1. Y ranges from 0 to 47. It is used only in multicolor graphics mode.

& VLN, Y1, Y2, X

VERTICAL LINE: This command is similar to the Apple low resolution graphics command: VLIN Y1, Y2 AT X. It draws a vertical line in the last selected color from the point (X, Y1) to the point (X, Y2). X ranges from 0 to 63. Y1 and Y2 range from 0 to 47, but Y2 must be greater than Y1. It is used only in multicolor graphics mode.

6.0 COMMANDS AFFECTING THE SPRITE ATTRIBUTE TABLE

```
& SSPR, S, Y, X, [, P ] [, C ] [, S, Y, X ] [, P ] [, C ]
```

SET SPRITE: This command sets the attributes of sprite number S in the sprite attribute table. S is the sprite number 0 to 31. Y is the vertical position which has two ranges: 0 to 191 and 225 to 255 (see next paragraph). X is the horizontal position and ranges from 0 to 255. See the VDP Reference Manual, Section 3.7, for information on how the sprite X and Y positions are defined. P is the sprite pattern number which determines which of the 256 shapes defined in the sprite pattern generator table is to be given to sprite S. C is the color of the sprite and ranges from 0 to 15 or 128 to 143 depending on the early clock bit which is the most significant bit of C (see Section 3.7 of the VDP Reference Manual).

The vertical range of the sprites is divided into two ranges to allow for negative Y values. The range 225 to 255, which in two's complement notation is - 31 to -1, corresponds to the range of negative Y values. These negative values allow a sprite to bleed in from the top of the screen. Y values of 0 to 191 are the normal positive values.

There is a simple way to deal with these negative Y values in BASIC. It boils down to converting signed negative values, such as -31 or -1, to equivalent 8 bit two's complement expressions by merely adding 256. Example: -31 becomes -31 + 256 = 225. So in BASIC the range of Y values may extend from -31 to 191, providing 256 is added to all negative Y values before using the & SSPR or the & SY commands. This is easily done by the following line of code:

```
IF Y >  -32  AND Y < 0 THEN Y = Y + 256
```

The X position may also have negative values from -32 to -1. This is done by setting the early clock bit. See the VDP Reference Manual, Section 3.7 for more details. Setting the early clock bit shifts the X position 32 pixels to the left. This allows sprites to bleed in to the left edge of the active display area of the screen. Using the early clock bit extends the X range from -32 to 255.

Handling negative X values can easily be done in BASIC. For all X values in the range -32 to -1 add 32 to the value before sending it to the sprite attribute table, and set the early clock bit in the color byte of the sprite attribute table. To set the early clock bit add 128 to the color value ONLY if the color is less that 16 (if it is more that 16, then it is assumed that the early clock bit is already set) and send the new value to the sprite attribute table using the & SSPR or & SC commands. If the color is not already known, it can can be read from the sprite attribute table by using the & GSPR or & GC commands.

For normal values of X, from 0 to 255, care must be taken to make sure the early clock bit is reset (to 0). If this is not done, the sprite may be shifted 32 pixels to the left of the desired X position if the early clock bit was previously set. Resetting the early clock bit requires subtracting 128 from the color value, ONLY if the color value is 128 or more (if it is less than 128, then the early clock bit is already reset), and sending the new value to the sprite attribute table using the & SSPR or & SC commands.

The variable P must be an integer multiple of 4 (0, 4, 8, 12, etc.) for size 1 sprites. This is because every four consecutive sprite patterns in the sprite pattern generator table are required to define a size 1 sprite. Therefore patterns 0 to 3 define one size 1 sprite, patterns 4 to 7 define another size 1 sprite, etc. The sprite pattern number of the first pattern of each of these sets of four patterns is an integer multiple of 4. If any other value is specified for P, the VDP will round it off to the next lowest integer multiple of 4. For Size 0 sprites P may have any integer value from 0 to 255.

The & SSPR command, or the & SY command, may be used to place the sprite terminator value 208 ($DO) into the vertical position byte of the first unwanted sprite of the sprite attribute table. This is necessary so that sprites not used will no show up as garbage on the screen.

Note: Variables S, Y and X must be specified as a set. P and C are optional. If however, P and C are specified, a second COMPLETE set of S, Y and X variables may be specified. The second P and C are again optional. This sequence can repeat for as long as the allowed BASIC line length is not exceeded.

& GSPR, S, Y, X, P, C

GET SPRITE: This command gets the sprite attributes from the sprite attribute table. It returns the vertical and horizontal positions of sprite S, the sprite pattern number P and the color into Y, X, P and C respectively. See the & SSPR command for a discussion of the meanings of the X and Y values. Care must be taken when using this command to read the X position of a sprite. If the early clock bit is set (C is 128 or greater) then the true position of the sprite is acutally 32 pixels less than the value X returned. Also Y values in the range 225 to 255 actually are negative Y values in the range -31 to -1.

```
& SX, S, X  [, ... ]
```

SET X: This command places sprite S at the horizontal position given by X. The vertical position Y, the sprite pattern number P, and the sprite color C remain unchanged, and are assumed to have been previously defined. This command places the value X into the horizontal position byte for sprite number S in the sprite attribute table. See the & SSPR command for further details on the variable X.

```
& GX, S, X
```

GET X: This command returns the X position of sprite S in the variable X. It reads the sprite attribute table to get the horizontal position of the sprite. See the & GSPR command for further details on the variable X.

```
& SY, S, Y  [, ... ]
```

SET Y: This command places sprite S at the vertical position given by Y. The horizontal postition X, the sprite pattern number P, and the sprite color C remain unchanged, and are assumed to have been previously defined. This command places the value Y into the vertical position byte for sprite number S in the sprite attribute table. See the & SSPR command for further details on the variable Y. This command may be used to clear (deactivate) all sprites from the screen beyond sprite number S. The command:

```
& SY, S, 208
```

deactivates all sprites from sprite S to sprite 31. The command:

```
& SY, 0, 208
```

may be used to deactivate all sprites. The vertical position value, 208, is the sprite attribute table terminator byte. Processing of all sprites in the sprite attribute table beyond the 208 is disabled.

```
& GY, S, Y
```

GET Y: This command returns the Y position of sprite S in the variable Y. It reads the sprite attribute table to get the vertical position of the sprite. See the & GSPR command for further details on the variable Y.

& SP, S, P [, ...]

SET PATTERN: This command sets the shape of sprite S to the pattern number in the sprite pattern generator table given by P. The horizontal position X, the vertical position Y, and the sprite color C remain unchanged, and are assumed to have been previously defined. This command places the value P into the sprite name byte for sprite number S in the Sprite Attribute Table. See the command & SSPR for further details on the variable P.

& GP, S, P

GET PATTERN: This command returns the sprite pattern number of sprite S in the variable P. It reads the sprite attribute table to get the pattern number of the sprite.

& SC, S, C [, ...]

SET COLOR: This command sets the color of sprite S to the color represented by C. The horizontal position X, the vertical position Y, and the sprite pattern P remain unchanged, and are assumed to have been previously defined. This command places the value C into the sprite color byte for sprite number S in the sprite atribute table. The early clock bit of the sprite color byte may also be set with this command. See & SSPR for more details on this bit and its significance.

& GC, S, C

GET COLOR: This command returns the color of sprite S in the variable C. It reads the sprite attribute table to get the color of the sprite. The value of the early clock bit is also returned. If the value of C is 128 or greater, then the early clock bit is set, and the X position of the sprite is actually 32 pixels to the left of the value that would be returned by the & GX and & GSPR commands. See & SSPR and & GSPR for more details on the early clock bit.

6.0 COMMANDS AFFECTING THE SPRITE PATTERN GENERATOR TABLE

& DSPR, P, P0, P1, P2, P3, P4, P5, P6, P7 [, ...]

DEFINE SPRITE: This command is similar to the & DTIL command. It is used to define the shape of sprite pattern P. P0 defines the top row (row 0) of the 8 x 8 sprite pattern, P1 defines the second row (row 1), P2 defines the third row (row 2), etc. This command writes the values of the eight variables P0 to P7 into the eight sequential addresses, corresponding to sprite pattern P, in the sprite pattern generator table. P and P0 to P7 all range from 0 to 255.

Size 1 sprites require that four separate and sequential 8 x 8 sprites (size 0) be defined, either by using four seperate & DSPR commands, or by using the optional repeat variable specification. The first pattern number of this block of four 8 x 8 sprite patterns must be an integer multiple of 4 (0, 4, 8, 12, etc.). See the VDP Reference Manual, Sections 3.7 and 9.0 for further details on sprite pattern definition.

```
& RSPR, P, P0, P1, P2, P3, P4, P5, P6, P7  [, ... ]
```

READ SPRITE: This command is used to read the eight bytes of data that define the 8 x 8 sprite pattern P. It reads the eight bytes from the sprite pattern generator table, and returns their values in the eight variables P0 to P7. The meanings and ranges of the variables is the same as in the & DSPR command.

8.0 PROGRAMMABLE SOUND GENERATOR COMMANDS

```
& SOFF
```

SOUND OFF: This command turns off all sounds generated by the PSG. All the noise and tone bits of the mixer register are disabled, and the volume registers are all set to 0.

```
& VOLA, V
& VOLB, V
& VOLC, V
```

VOLUME: These commands are used to set the volume levels of PSG channels A, B, and C respectively to the value V. V ranges from 0 (volume off) to 15 (maximum volume) for manual volume control. When using the automatic envelope control feature of the PSG, V must be set to 16.

The PSG' s internal amplifier's gain for any one of the three channels depends upon the volume settings of the other two channels. For example if channel A is at maximum volume and channels B and C are at 0 volume, channel A will have a relatively large volume level. Now if you increase the volume level of any of the other two channel, the volume of channel A will be diminished.

```
& TNA, P [, V ]  [, P ]  [, V ]  [, P ]  [, V ]
& TNB, P [, V ]  [, P ]  [, V ]  [, P ]  [, V ]
& TNC, P [, V ]  [, P ]  [, V ]  [, P ]  [, V ]
```

TONE: Each of these commands is used to play one, two, or three tones on the selected channel(s). They set the fine and coarse tune registers of the given channel to the 12-bit tone period value P (1 - 4095), and optionally set the volume register of the channel to the value V (0 - 16). The tone bit in the PSG mixer register is enabled. The range of tone frequencies is from around 120 KHz (P = 1) to about 30 Hz (P = 4095). The value P represents a period value (the inverse of frequency) and thus the higher the value of P the lower the tone frequency, and vice versa.

Each command can be used to play 1, 2 or 3 tones merely by specifying the optional parameters. If more than one period value is given the next channel (in cyclic order) is assigned the specified period.

The command:

```
& TNA, P
```

plays a tone on channel A of period P at whatever volume for channel A was last specified. The command:

```
& TNB, P, V
```

plays a tone on channel B of period P at volume V. The command:

```
& TNA, P1, V1, P2
```

plays two tones simultaneously, one on channel A of period P1 and volume V1 and the other on channel B of period P2 at whatever volume was last specified for channel B. The command:

```
& TNB, P1, V1, P2, V2
```

plays two tones simultaneously, one on channel B of period P1 and volume V1 and the other on channel C of period P2 and volume V2. The command:

```
& TNC, P1, V1, P2, V2
```

plays two tones simultaneously, one on channel C of period P1 and volume V1 and the other on channel A of period P2 and volume V2. Note how the channels follow each other in a cyclic order. The commands:

& TNA, P1, V1, P2, V2, P3, V3
& TNB, P1, V1, P2, V2, P3, V3
& TNC, P1, V1, P2, V2, P3, V3

all are essentially identical in that three tones are played simultaneously. The first of period P1 and volume V1, the second of period P2 and volume V2, and the third of period P3 and volume V3. The only difference in each of these commands is in which channel plays which note, but the results are the same.

& NSA, NP, [, V]
& NSB, NP, [, V]
& NSC, NP, [, V]

NOISE: Each of these commands sets the PSG noise period register to the noise period value NP (1 - 31) and enables the noise bit for the appropriate channel in the PSG mixer register. Optionally the volume register for the channel is set to V (0 - 15).

& ENVP, P

ENVELOPE PERIOD: This command is used to set the envelope period for the automatic envelope control feature of the PSG. It sets the fine and coarse envelope period registers of the PSG to the 16-bit value P (1 - 65535). See the PSG Reference Manual, under Automatic Envelope Control, for more information on the envelope periods.

& ENVT, T

ENVELOPE TYPE: This command is used to set the envelope type register of the PSG to the value T (0-15) for use with the automatic envelope control feature of the PSG. See the PSG Reference section, under Automatic Envelope Control, for more information on the envelope types.

& FILT, F

FILTER: This command is used to select the filter type to be used to modify the sound output of the PSG. F ranges from 0 to 15. 0, 4, 8, and 12 are high pass; 1, 5, 9, and 13 are low pass; 2, 6, 10, and 14 are band pass; and 3, 7, 11, and 15 are notch pass. See the PSG Reference manual, under Programmable Filters, for more information.

& ETNA, P [, P] [, P]
& ETNB, P [, P] [, P]
& ETNC, P [, P] [, P]

ENVELOPE TONE: These commands are used to create tones with automatic envelope control to simulate sounds like chimes, piano, etc. Three things must be done before using these commands: the volume of the selected channel must be set to 16, and both the envelope period and envelope type must be defined. Once these things are done, they need not be done again. These commands enable the tone bits in the PSG mixer register for the selected channel(s), set the tone period register(s) to the value(s) P, and trigger the PSG's automatic envelope control circuitry by sending the previously-defined envelope type value T to the PSG's envelope type register. Two or three note chords may be played by specifying additional period values for the other tone channels (see the & TNA, & TNB, & TNC commands).

& ENSA, NP
& ENSB, NP
& ENSC, NP

ENVELOPE NOISE: These commands are used to create sound effects featuring noise and automatic envelope control, such as explosion, drum, or cymbal sounds. Three things must be done before using these commands: the volume of the selected channel must be set to 16, and both the envelope period and envelope type must be defined. Once these are done, they need not be done again. These commands enable the noise bits in the PSG mixer register for the selected channel, set the noise period register to the value NP, and trigger the PSG's automatic envelope control circuitry by sending the previously-defined envelope type value T to the PSG's envelope type register.

& MXR, MX

MIXER: This command is used to enable or disable the tone and noise bits of the mixer control register (7) of the PSG. See the PSG Reference section for more details. It is useful for shutting off the tone or noise, or both, from a channel, without setting the volume of that channel to 0, of using the & SOFF command. Due to the way the PSG's internal amplifier works, the volume levels of each channel are not totally independent of the volume levels of the other channels. Changing the volume level of one channel can affect the volume level of another channel. Therefore when multiple channels are in use, it is not advised to set a channel's volume to 0 in order to shut off that channel, but to rather disable the channel's tone or noise bits in the mixer register.

9.0 MISCELLANEOUS COMMANDS

& LD, S, D, L

LOAD: This command is used to load a block of data into the video RAM (VRAM) from the Apple's main memory. Typically this is used when a file containing VRAM table data is loaded from disk into Apple memory, and then this data is in turn loaded into VRAM. S is the source address in Apple memory, D is the destination address in VRAM, and L is the length of the data block, i.e. the number of bytes to transfer.

& ULD, S, D, L

UNLOAD: This command is used to unload a block of data from the video RAM (VRAM) into the Apple's memory., S is the source address in VRAM, D is the Apple memory destination address, and L is the number of bytes to unload, i.e. the length of the block. This command would typically be used to save a VRAM table to disk.

==

TMS 9918A VIDEO DISPLAY PROCESSOR

TECHNICAL REFERENCE MANUAL

==

1.0 BASIC FEATURES

The Texas Instruments TMS 9918A Video Display Processor's (VDP) main features are:

- 32 sprites for easy animation
- 16 colors (15 true colors plus transparent)
- 35 graphics planes for pseudo 3-D effects
- two 256H X 192V resolution graphics modes with 16 colors
- one 64H x 48V resolution graphics mode with 16 colors
- 24 row by 40 column text mode, user-defined character set
- table-driven graphics for rapid animation and color changes
- status register providing information on sprite collisions
- 60 Hz interrupt during vertical blank period
- NTSC-compatible, non-interlaced, composite video output

Most of these features will be explained in detail throughout this manual.

2.0 VDP REGISTERS

The VDP contains eight write-only registers and one read-only status register. The eight write-only registers are used to enable or disable the various modes and features of the VDP, to allocate the 16 K of video RAM (VRAM) to the various tables which are used to define the video screen image, and to set the text / backdrop color(s). The read-only status register provides information on the status of interrupts, sprite collision, and fifth sprite number. The status register also is used to reset the VDP's internal logic for data transfers to and from the VDP. All of these features will be explained in more detail below.

Writing data to the eight write only registers consists of two byte transfers. The first byte written must be the data byte and the second byte written is the register number. Reading the status register is only a one byte transfer. See section 4.3 and 4.4 for more details.

2.1 VDP WRITE-ONLY REGISTERS

The eight write-only registers are divided into three main types. Registers 0 and 1 control which of the various display modes and features of the VDP are enabled or disabled, registers 2 to 6 control video RAM (VRAM) allocation to the various tables which define the screen image, and register 7 controls the backdrop color and the text color. Each of these register is described below in more detail. Figure 1 shows the various bits of the VDP registers.

REGISTER	MSB							LSB
0	0	0	0	0	0	0	M3	EV
1	4/16K	BLANK	IE	M1	M2	0	SIZE	MAG
2	0	0	0	0	NAME TABLE BASE ADDRESS			
3	COLOR TABLE BASE ADDRESS							
4	0	0	0	0	0	PATTERN GENERATOR BASE ADDRESS		
5	0	SPRITE ATTRIBUTE TABLE BASE ADDRESS						
6	0	0	0	0	0	SPRITE PATTERN GENERATOR BASE ADDRESS		
7	TEXT COLOR$_1$				TEXT COLOR$_0$/BACKDROP COLOR			
STATUS (READ-ONLY)	F	5S	C	FIFTH SPRITE NUMBER				

FIGURE 1 - VDP REGISTERS

2.1.1 REGISTER 0

Register 0 contains two control bits only. All other bits must be set to 0.

bit 0 - EV, external video enable / disable
bit 1 - M3 mode

Bits 2 to 7 are reserved for future use and must be set to 0. Strange things may happen if these bits are not 0.

The external video enable / disable bit is used to allow an external video signal to become the external video plane of the VDP's multiplanar graphics structure (see section 3.1). This is usually used to cascade two or more VDPs, or to overlay the VDP's video image on top of an external video signal. This is explained in section 8.0 in greater detail.

The mode bit M3 is used in conjunction with the two other mode bits M1 and M2 in register 1. These three mode bits are used to select the one of the four display modes of the VDP. See next section.

2.1.2 REGISTER 1

Register 1 contains seven control bits. The unused bit, bit 2, is reserved for future use and must be set to 0.

bit 0 - MAG, sprite magnification
bit 1 - SIZE, sprite size
bit 2 - reserved, set to 0
bit 3 - M2 mode
bit 4 - M1 mode
bit 5 - IE, interrupt enable
bit 6 - BLANK, blanks screen
bit 7 - 4/16 K RAM selection

The MAG bit controls the magnification of the sprites. When set to 0 the sprites are at magnification 0, and when set to 1 the sprites are at magnification 1.

The SIZE bit controls the size of the sprite patterns. When set to 0 the sprites are 8 x 8 pixels in size, and when set to 1 the sprites are 16 x 16 pixels in size.

The mode bits M1 and M2, along with M3 of register 0 determine the display mode of the VDP. Refer to Table 1 for valid combinations of these mode bits. All other combinations of the mode bits will lead to invalid display modes.

The IE bit is used to enable or disable the generation of VDP interrupts during the vertical blank period. These interrupts occur approximately every 1/60 th of a second.

M1	M2	M3	DISPLAY MODE
0	0	0	graphics mode 1
0	0	1	graphics mode 2
0	1	0	multicolor mode
1	0	0	text mode

TABLE 1: MODE BITS AND DISPLAY MODES

The BLANK bit enables or disables the active display area of the screen. When set to 0 the active display area is disabled and only the backdrop / border color is seen on the screen. When set to 1 the active display area is enabled. The active display area consists of the pattern plane and the 32 sprite planes.

The 4/16 K selection bit is used to select either 4 K or 16 K RAMs. When 0 4 K RAMs are selected and when 1 16 K RAMs are selected. This bit should always be set to 1, as 16 K RAMs are used.

2.1.3 REGISTER 2 - NAME TABLE BASE POINTER

Register 2 contains the name table base pointer and defines the base address of the name table in VRAM. The 4 high order bits of this register must be 0. The 4 low order bits of this register form the upper 4 bits of the 14 bit name table address. The range of values allowed in this register is 0 to 15, hence there are 16 possible locations for the name table in VRAM. The name table must begin on a 1 K ($400) boundary and is 768 bytes long for all graphics modes and 960 bytes long for text mode. The base address of the name table can be calculated by the formula:

(name table base) = (register 2) * $400

2.1.4 REGISTER 3 - COLOR TABLE BASE POINTER

Register 3 contains the color table base pointer and defines the base address of the color table in VRAM. The range of values is from 0 to 255, thus there are 256 possible locations for the color table. The color table is used only in graphics modes 1 and 2 and is not used in multicolor or text modes. The meaning of the bits in this register depends on whether the VDP is in graphics mode 1 or graphics mode 2.

In graphics mode 1 the color table must begin on a 64 ($40) byte boundary, and any of the possible 256 different color table locations may be used. Only the first 32 bytes of the color table are used and the last 32 bytes are wasted. The contents of register 3 form the upper 8 bits of the 14 bit color table address. The color table base address of the graphics mode 1 color table can then be calculated by the formula:

(color table base) = (register 3) * $40

In graphics mode 2, however, the color table must begin on an 8 K ($2000) boundary, and thus there are only two possible color table base addresses possible: $0000 and $2000. The color table is 6 K bytes long ($1800). To select a color table base address of $0000, register 3 must contain 127 ($7F). To select the color table base address of $2000, register 3 must contain 255 ($FF). For graphics mode 1 the register 3 values would be 0 and 128 ($80) respectively for the same color table base addresses. The difference is due to the fact that the 7 least significant bits of register 3 must always be set to 1 to select the graphics mode 2 color table base location. The highest order bit determines which of the two possible locations is used. The base address of the graphics mode 2 color table can be calculated by the formula:

(color table base) =[(register 3) AND $80] * $40

If register 3 does not contain either 127 ($7F) or 255 ($FF), then the graphics mode 2 display will not function properly.

2.1.5 REGISTER 4 - PATTERN GENERATOR TABLE BASE POINTER

Register 4 contains the pattern generator table base pointer and defines the base address of the pattern generator table in VRAM. The 5 high order bits must be set to 0, hence the range of legal values is from 0 to 7. The pattern generator table is used in all four display modes. The meaning of the bits of this register depends upon which display mode the VDP is in.

In graphics mode 1, multicolor graphics mode, and text mode there are eight possible locations for the pattern generator table. The pattern generator table must begin on a 2 K ($800) boundary and is 2 K bytes long. The contents of this register form the 3 high order bits of the 14 bit pattern generator table address. The pattern generator table base address can be calculated by the formula:

(pattern generator table base) = (register 4) * $800

In graphics mode 2, however, the pattern generator table must begin on an 8 K ($2000) boundary, and thus there are only two possible pattern generator table base addresses possible: $0000 and $2000. The pattern generator table is 6 K bytes long ($1800). To select a pattern generator table base address of $0000, register 4 must contain 3. To select the pattern generator table base address of $2000, register 4 must contain 7. For graphics mode 1 the register 4 values would be 0 and 4 respectively for the same pattern genrator table base addresses. The difference is due to the fact that the 2 least significant bits of register 4 must always be set to 1 to select the graphics mode 2 pattern generator table base location. The highest order bit determines which of the two possible locations is used. The base address of the graphics mode 2 pattern generator table can be calculated by the formula:

(pattern generator table base) =[(register 4) AND $04] * $800

If register 4 does not contain either 3 or 7, then the graphics mode 2 display will not function properly.

2.1.6 REGISTER 5 - SPRITE ATTRIBUTE TABLE BASE POINTER

Register 5 contains the sprite attribute table base pointer and defines the base address of the sprite attribute table in VRAM. The most significant bit of this register must be set to 0, so the legal range is from 0 to 127. There can be 128 different locations for the sprite attribute table. Each table is 128 ($80) bytes long, and must begin on a 128 byte boundary. The contents of this register form the upper 7 bits of the 14 bit sprite attribute table address. The base address of the sprite attribute table can be found from the formula:

(sprite attribute table base) = (register 5) * $80·

2.1.7 REGISTER 6 - SPRITE PATTERN GENERATOR TABLE BASE POINTER

Register 6 contains the sprite pattern generator table base pointer and defines the base address of the sprite pattern generator table in VRAM. The 5 high order bits must be set to 0, hence the range of legal values is from 0 to 7. There are eight possible locations for the sprite pattern generator table. The sprite pattern generator table must begin on a 2 K ($800) boundary and is 2 K bytes long. The contents of this register form the 3 high order bits of the 14 bit sprite pattern generator table address. The sprite pattern generator table base address can be calculated by the formula:

(sprite pattern generator table base) = (register 6) * $800

2.1.8 REGISTER 7 - BACKDROP / TEXT COLOR

The low 4 bits of register 7 define the backdrop / border color for all display modes. The high 4 bits are used only in text mode and define the color of the text characters.

2.2 VDP STATUS REGISTER

The VDP has a single 8 bit status register which contains the interrupt pending flag, the sprite coincidence flag, the fifth sprite flag, and the fifth sprite number, if one exists. Refer to Figure 1 for the format of the status register.

The status register may be read at any time to test the F, C, and 5S status bits. Reading the status register will reset the interrupt pending flag F to 0. Asynchronous reads will cause the interrupt pending flag F to be reset to 0, and therefore missed.

Consequently, the status register should be read only when the VDP interrupt is pending.

2.2.1 INTERRUPT FLAG (FRAME FLAG) F

This flag is set to 1 at the end of the raster scan of the last line of the active display, i.e. at the end of each frame of the video display. It is reset to 0 after the status register is read or when the VDP is externally reset. If the VDP interrupt enable bit IE of VDP register 1 is set to 1, then the VDP will generate an interrupt whenever the F flag is set. The interrupt pin is tied to the 6502's IRQ line, and thus the interrupt is maskable.

Note that the status register must be read frame by frame (after each interrupt) in order to reset the F bit to 0 so that the next interrupt can be received. If this is not done, no further interrupts will be generated until the status register is read.

2.2.2 COINCIDENCE FLAG C

The C status flag is set to 1 if two or more sprites collide. Collision occurs if any two sprites on the screen have one or more overlapping pixels. Transparent areas of colored sprites, transparent sprites, and sprites that are partially or completely off the screen are all considered. Sprites beyond the sprite attribute table terminator byte 208 ($D0) are not considered. The C flag is reset to 0 after the status register is read, or when the VDP is externally reset. The status register should be read upon power up to insure that the C flag is reset.

The VDP checks each pixel position for coincidence during the generation of the pixel regardless of where it is located on the screen. This occurs every 1 / 60 th of a second. Thus, when moving sprites more than one pixel position during these intervals, it is possible for sprites to have multiple pixels overlapping or even to have passed completely over one another when the VDP checks for coincidence.

2.2.3 FIFTH SPRITE FLAG 5S AND NUMBER

The 5S status flag is set to 1 whenever there are five or more sprites on a horizontal line and the F flag is equal to 0. The 5S status flag is reset to 0 after the status register is read or the VDP is externally reset. The number of the fifth sprite is placed into the lower 5 bits of the status register whenever the 5S flag is set and is valid only when the 5S flag is equal to 1. The setting of the 5S flag does not cause an interrupt.

3.0 VIDEO DISPLAY MODES AND FEATURES

The VDP has four video display modes: graphics mode 1, graphics mode 2, multicolor graphics mode, and text mode. The three graphics modes have a unique multiplanar structure allowing for pseudo 3-dimensional effects. The graphics modes also have another special feature: sprites. Sprites are special patterns that can be easily moved and are designed specifically for animation. The VDP allows up to 32 sprites on the screen at any given time. Sprites are not available for use in the text mode.

The maximum resolution of the VDP display modes is 256H x 192V. This resolution is available in graphics mode 1 and graphics mode 2. The multicolor graphics mode is a low resolution mode of only 64H x 48V resolution. The text mode has 240H x 192V resolution. In all three graphics modes 16 colors are simultaneously available. These colors and their values are listed in Table 2.

COLOR HEX	COLOR	TMS9918A	
		LUMINANCE (DC) VALUE	CHROMINANCE (AC VALUE)
0	TRANSPARENT	0.00	-
1	BLACK	0.00	-
2	MEDIUM GREEN	.53	.53
3	LIGHT GREEN	.67	.40
4	DARK BLUE	.40	.60
5	LIGHT BLUE	.53	.53
6	DARK RED	.47	.47
7	CYAN	.67	.60
8	MEDIUM RED	.53	.60
9	LIGHT RED	.67	.60
A	DARK YELLOW	.73	.47
B	LIGHT YELLOW	.80	.33
C	DARK GREEN	.46	.47
D	MAGENTA	.53	.40
E	GRAY	.80	-
F	WHITE	1.00	-
-	BLACK LEVEL	0.00	-
-	COLOR BURST	0.00	.40
-	SYNC LEVEL	-0.40	-
-	EXTERNAL VIDEO	-	-
-	LEVEL	-	-

TABLE 2: COLOR ASSIGNMENTS

In the text mode any two of the set of 15 true colors may be selected, giving 15 x 14 = 210 different color combinations.

Depending on the quality of the TV or monitor, not all color combinations are legible.

3.1 MULTIPLANAR GRAPHICS STRUCTURE

The VDP's graphics image can best be envisioned as a set of 35 display planes arranged in a stack. Figure 2 shows this structure. The planes closest to the viewer have higher priority. When two objects on different planes occupy the same spot on the screen, the object on the higher priority plane will show at that point. For an object on a given plane to show through, all higher priority planes must be transparent at those points occupied by the object.

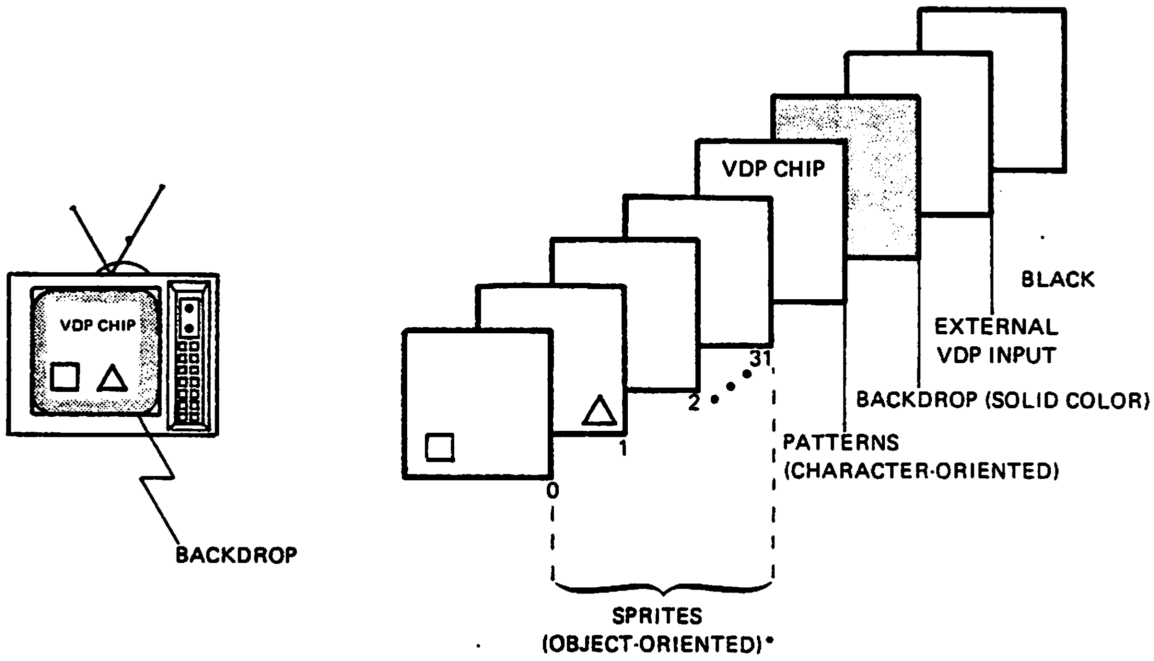


FIGURE 2 - VDP DISPLAY PLANES

The first 32 planes nearest the viewer are the sprite planes. See Figure 3. Each sprite plane can contain one sprite of a single color. The sprite planes are transparent in all areas outside of the sprite itself. All or part of any sprite may be transparent. Whenever a pixel of a sprite plane is transparent, the color of the next lower priority plane can be seen at that point. If, however, the pixel of the sprite plane is colored, the color of the pixel of the next lower priority plane is replaced by the sprite color. The sprites can cover either an 8 x 8 pixel area, a 16 x 16 pixel area, or a 32 x 32 pixel area.

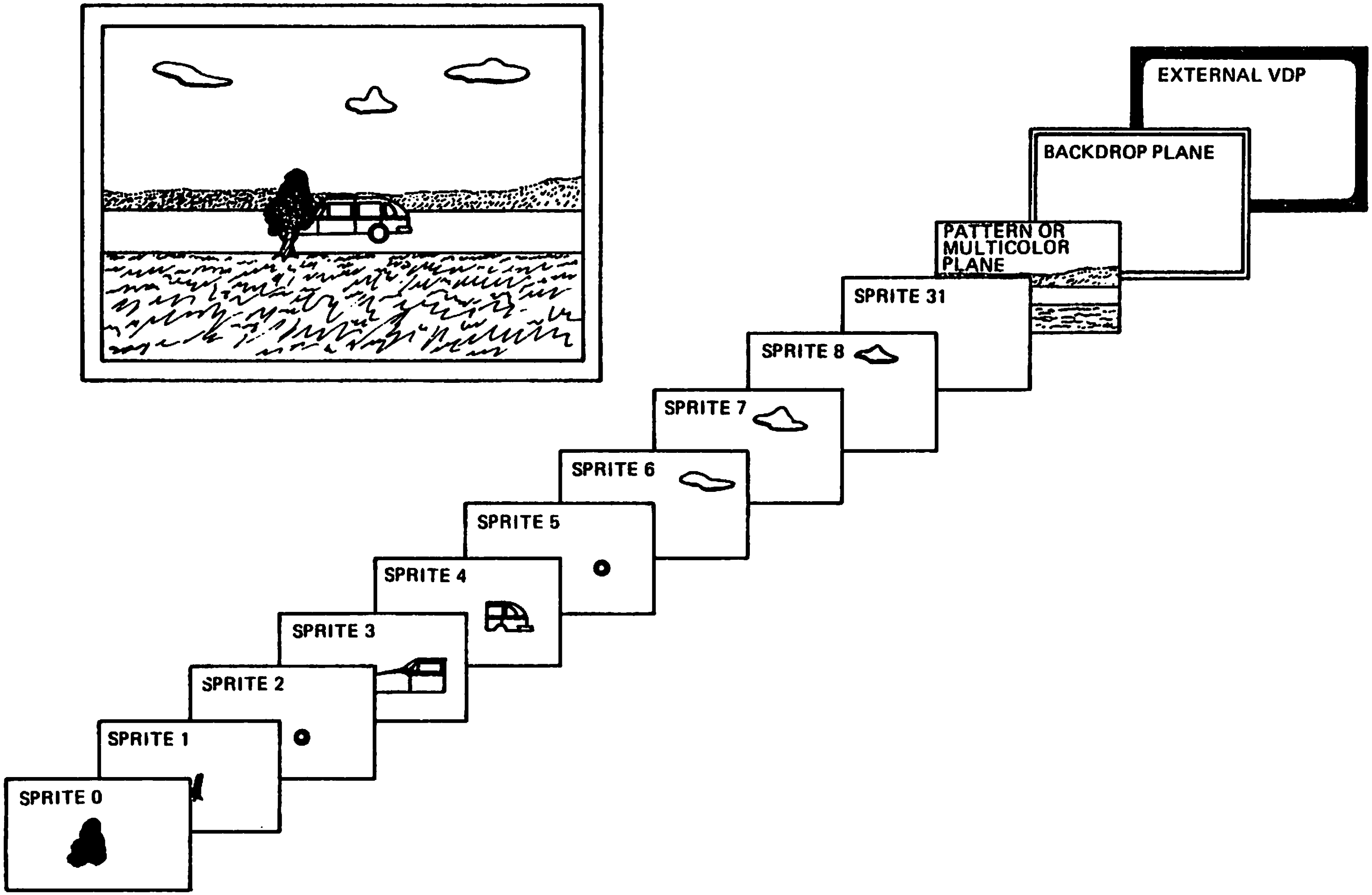


FIGURE 3 - DEFINITIONS OF VDP GRAPHICS PLANES

Directly behind the sprite planes is the pattern plane, which is the normal graphics plane of most computer graphics systems. The pattern plane can be filled entirely with multicolored patterns. The three different graphics modes differ only in the resolutions and color restrictions of the pattern plane; the sprite planes are identical in all three graphics modes. The pattern plane is also used for the text mode to display the user-defined text characters. If any pixel on the pattern plane is transparent, then the backdrop color will show through.

The 34 th plane is the backdrop color plane. It is a solid color and is larger than all the other planes. It therefore forms a colored border around the active display area. The color of the backdrop plane is determined by the low 4 bits of VDP register 7.

The lowest priority plane is the external video plane. When the EV bit in VDP register 0 is enabled, the external video signal forms this plane's image. With the appropriate hardware, the other VDP planes can be superimposed upon this external video plane. See Section 8.0 for more details.

3.2 TABLE DRIVEN GRAPHICS

Unlike the bit-mapped graphics of most computers (the Apple II included) the VDP's graphics are table driven. The 16 K of video RAM (VRAM) is subdivided into various tables which define the display image. Animation is acheived by altering the contents of these tables in real time. There is usually room for several alternate sets of tables in VRAM. Extremely rapid animation can be acheived merely by switching from one set of tables to another. This is done by changing the contents of VDP registers 2 to 6, which define the base addresses of the various tables in VRAM. In graphics mode 2 the tables may be set up in such a way as to provide bit-mapped graphics.

3.3 GRAPHICS MODE 1

Since the only difference between the three graphics modes is in the pattern plane, only the graphics mode 1 pattern plane is described below. Details on the sprites and the 32 sprite planes are explained separately in section 3.7.

The VDP is in graphics mode 1 when mode bits M1, M2 and M3 are all 0. In this mode the pattern plane is divided into a grid of 32 columns and 24 rows of character cells (also called tile positions) as shown in Figure 4. There are 32 x 24 = 768 different character cells. Each one of these character cells is further subdivided into an 8 x 8 pixel pattern. This, therefore, provides for a resolution of 32 x 8 = 256 horizontal pixels and 24 x 8 = 192 vertical pixels. The character cells are numbered, as shown in Figure 4, from 0 to 767.

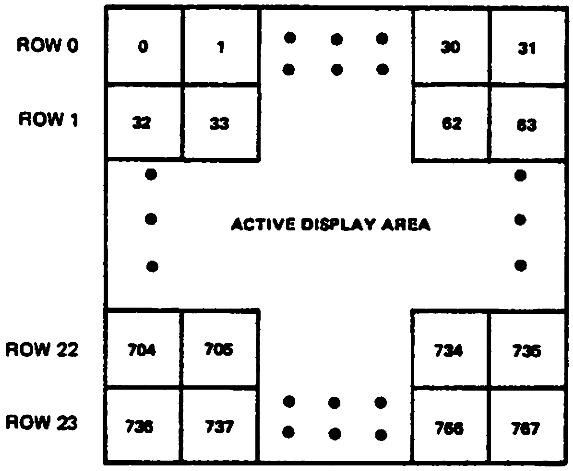


FIGURE 4 - GRAPHICS MODE 1 PATTERN PLANE MAPPING

There is a block of memory addresses in VRAM called the 'name table'. Each address in the name table corresponds to a particular character cell in the pattern grid. Since there are 768 character cells, the name table is 768 bytes long.

The starting address of the name table is determined by VDP register 2. The name table must begin on a 1 K boundary. There are 16 possible starting addresses for the name table. The absolute address of the name table locations will vary, depending on which of the 16 name table base addresses is chosen. See Section 2.1.3. Instead of absolute addresses, we will refer to relative addresses, such as entry 0 of the name table, to represent the first address of the name table.

The mapping of name table addresses to character cells is simple and straight forward. Entry 0 of the name table corresponds to character cell 0 of the pattern grid, entry 1 of the name table corresponds to character cell 1 of the pattern grid, etc. In general, entry N of the name table corresponds to character cell N of the pattern grid.

Into each of the 768 character cells can be 'placed' a specific 8 x 8 pixel pattern. The particular pattern which shows up on the screen in a given character cell depends upon the contents of the corresponding name table entry. In graphics mode 1 there can be up to 256 different, user-defined, 8 x 8 pixel patterns, called tile patterns. These tile patterns are numbered from 0 to 255. Writing the number P (0 - 255) into a given name table location will make tile pattern P appear on the screen in the corresponding character cell. Since there are only 256 different patterns, each of the 768 character cell locations can not have a unique pattern. This can only be done in graphics mode 2.

The 256 tile patterns are defined in a block of VRAM addresses, called the pattern generator table. Each tile pattern is defined by 8 bytes of data as shown if Figure 5. The first byte defines the first row of 8 pixels of the tile pattern, the second byte defines the second row of 8 pixels, etc. The most significant bit (bit 7) of each of the 8 bytes defines the first (rightmost) column of the tile pattern, bit 6 of each of the 8 bytes defines the second column, and so forth. Tile pattern 0 is defined by the first 8 bytes in the pattern generator table (bytes 0 to 7), tile pattern 1 is defined by the second 8 bytes in the pattern generator table (bytes 8 to 15), and so forth.

ROW/BYTE	COLUMN (PATTERN)						BIT (PATTERN DEFINITION)							
	0	1	2	3	4	5	0	1	2	3	4	5	6	7
0		C	C	C	C	C	0	1	1	1	1	1	0	0
1						C	0	0	0	0	0	1	0	0
2						C	0	0	0	0	0	1	0	0
3			C	C	C	C	0	0	1	1	1	1	0	0
4						C	0	0	0	0	0	1	0	0
5						C	0	0	0	0	0	1	0	0
6		C	C	C	C	C	0	1	1	1	1	1	0	0
7							0	0	0	0	0	0	0	0

NOTES: VDP register 7 entry: 71_{16}.
Color code 7 is cyan (signified above by 'C').
Color code 1 is black (signified above by a space).
Bit 0 is the most significant bit of each data byte.

FIGURE 5 - TILE PATTERN DEFINITION

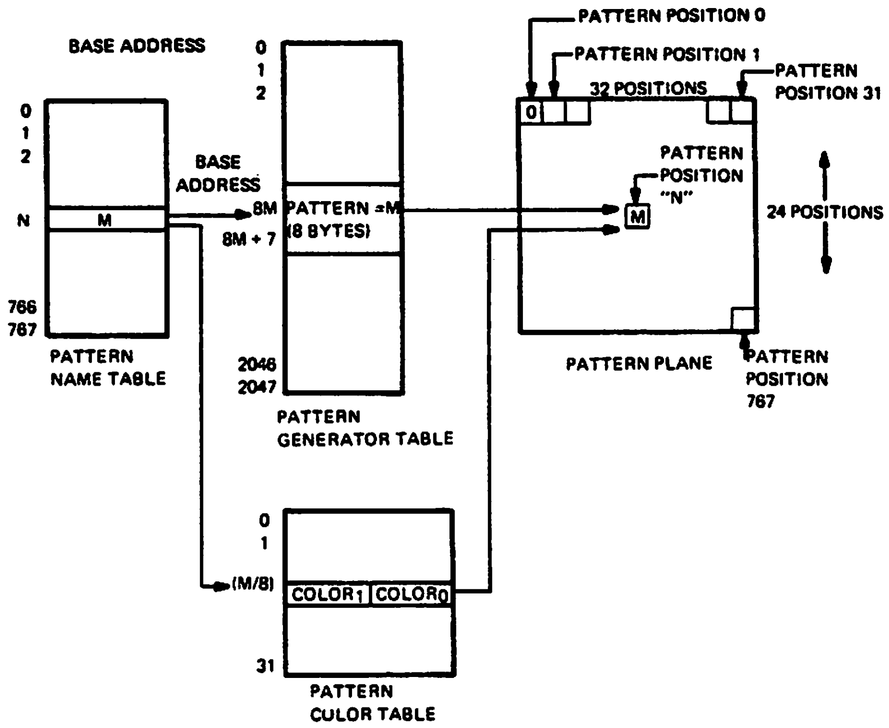
Since there can be up to 256 different tile patterns, each 8 bytes long, the pattern generator table can require a 256 x 8 = 2048 (2 K) block of VRAM. The pattern generator table must begin on a 2 K boundary. See Section 2.1.5. The pattern generator table base address is determined by VDP register 4. There are 8 possible starting locations for the pattern generator table.

Each of the 64 bits that define the tile pattern may be one of two user-defined colors. A 0 bit selects the '0' color for the corresponding pixel and a 1 bit selects the '1' color for the corresponding pixel. The actual colors represented by the '0' and '1' colors is determined by another VRAM table called the color table. There can be at most only two different colors in a graphics mode 1 tile pattern.

Each of the 256 different tile patterns can not have its own unique set of two colors. The tile patterns are grouped into sets of 8 patterns each, called color sets. There are 256 / 8 = 32 different color sets. Tile patterns 0 to 7 make up color set 0, tile patterns 8 to 15 make up color set 1, tile patterns 16 to 23 make up color set 2, and so forth. All tile patterns in a color set have the same '0' and '1' colors, so there can be at most only 32 unique tile pattern color combinations at a time.

The '0' and '1' colors of the 32 color sets are determined by the 32 single byte entries of the color table. The high 4 bits of each entry determines the '1' color for that color set and the low 4 bits determines the '0' color. Entry 0 of the color table determines the colors for color set 0, entry 1 determines the colors for color set 1, etc. See Figure 6.

The color table is 64 bytes long, but only the first 32 are used. The color table must begin on a 64 byte boundary in VRAM. The starting address of the color table is determined by VDP register 3. There are 256 possible starting locations for the color table. See Section 2.1.4.



Byte No.	Pattern No.
0	0..7
1	8..15
2	16..23
3	24..31
4	32..39
5	40..47
6	48..55
7	56..63
8	64..71
9	72..79
10	80..87
11	88..95
12	96..103
13	104..111
14	112..119
15	120..127
16	128..135
17	136..143
18	144..151
19	152..159
20	160..167
21	168..175
22	176..183
23	184..191
24	192..199
25	280..207
26	208..215
27	216..223
28	224..231
29	232..239
30	240..247
31	248-255

FIGURE 6 - GRAPHICS MODE 1 MAPPING

To summarize how graphics mode 1 works refer to Figure 6. Each of the 768 bytes of the name table correspond to a particular character cell of the pattern plane shown in Figure 4. The character cell is an 8 x 8 pixel area. The contents of each of the 768 name table entries points to a particular pattern in the pattern generator table. This pattern will appear on the screen in the corresponding character cell location. The colors of the 0 and 1 bits of this pattern are determined by the color table entry corresponding to the color set of which the pattern is a member.

For example: The contents of name table entry N, which corresponds to character cell N on the screen, contains the 8 bit value M. Tile pattern M defined by the 8 byte pattern generator table entry number M will appear on the screen at the character cell location N. Tile pattern M belongs to color set M / 8 (ignoring fractions). The M / 8 th entry of the color table determines the '0' and '1' colors of tile pattern M, as well as for the 7 other patterns in color set M / 8.

3.4 GRAPHICS MODE 2

The VDP is in graphics mode 2 when mode bits M1 and M2 of VDP register 1 are 0 and mode bit M3 of VDP register 0 is 1. Graphics mode 2 is similar to graphics mode 1, except that there can be up to 768 unique tile patterns, each with its own color specification independent of the other tile patterns. Also there are less color restrictions. Each row of the·tile pattern can have its own '0' and '1' colors, allowing a total of 16 possible colors within a tile pattern. Graphics mode 2 requires significantly more memory than any other display mode.

Like graphics mode 1, the graphics mode 2 pattern plane is divided into a grid of 32 horizontal by 24 vertical character cells as shown in Figure 4. Each character cell is further subdivided into an 8 x 8 pixel pattern area. The mapping of the character cells to the 768 bytes of the name table is exactly the same as in graphics mode 1.

Graphics mode 2 allows up to 768 unique tile patterns. Since each entry of the name table is only 8 bits wide, only tile patterns from 0 to 255 may be specified. This presents a problem. How does one specify tile pattern numbers greater than 255 for a particular character cell location?

This problem is solved by dividing the screen into three equal, horizontal blocks of 256 character cells each (3 x 256 = 768). Refer to Figure 7. Tile patterns 0 to 255 may be 'placed' only in the top third of the screen (character cell locations 0 to 255), tile patterns 256 to 511 may be 'placed' only in the middle third of the screen (character cell locations 256 to 511), and tile patterns 512 to 767 may be 'placed' only in the bottom third of the screen (character cell locations 512 to 767) only.

Similarly, the name table, the pattern generator table, and the color table are each divided into equal thirds. The first third corresponding to the top third of the screen, the second third corresponding to the middle third of the screen, and the last third corresponding to the bottom third of the screen.

For name table entries 256 to 511, the VDP's internal logic automatically adds 256 to the 8 bit tile pattern number. Thus if name table entry 256 contains the number P (0 - 255), tile pattern P + 256 will actually appear on the screen in character cell location 256. Similarly 512 is automatically added to the 8 bit pattern number for name table entries 512 to 767. Thus if name table entry 512 contains the number P (0 - 255) tile pattern P + 512 will actually appear on the screen in character cell location 512.

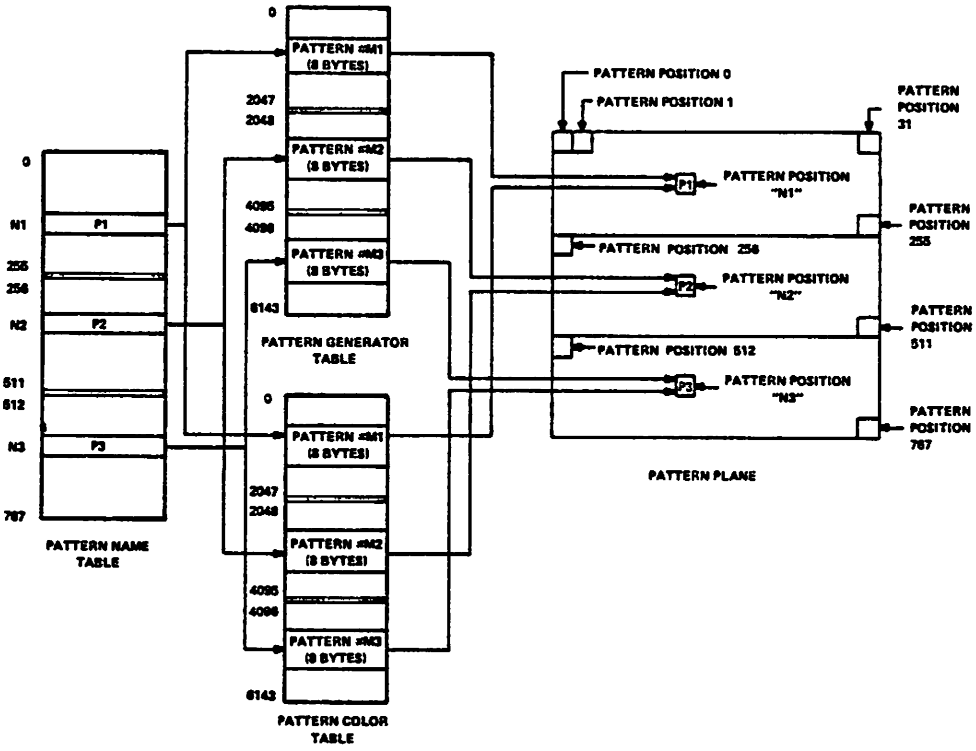


FIGURE 7 - GRAPHICS MODE 2 MAPPING

Since there can be up to 768 unique, user-defined patterns, the graphics mode 2 pattern generator table can be up to 8 x 768 = 6144 (6 K) bytes long. The graphics mode 2 pattern generator table must begin on an 8 K boundary. There can be only two possible starting addresses in VRAM for the pattern generator table. See Section 2.1.5.

The graphics mode 2 color table is essentially an extension of the pattern generator table. For each byte in the pattern generator table, there is a corresponding byte in the color table. Therefore the color table must be as long (up to 6 K) as the pattern generator table. For each tile pattern definition byte in the pattern generator table, there is a color definition byte (in the same relative location from the color table base address) in the color table. Tile pattern 0 is defined by bytes 0 to 7 of the pattern generator table, tile pattern 1 is defined by bytes 8 to 15, and so forth. Similary, the colors of tile pattern 0 are defined by bytes 0 to 7 of the color table, the colors of tile pattern 1 are defined by bytes 8 to 15 of the color table, and so forth. The '0' and '1' colors of each 8 pixel row of a tile pattern are defined by the low and high nibbles respectively of the corresponding color table byte. See Figure 8.

ROW	PATTERN GENERATOR TABLE ENTRY	PATTERN	PATTERN COLOR TABLE ENTRY		ROW
0	0 1 0 0 0 0 0 1	B 1 B B B B B 1	1 (BLACK)	B (LT. YELLOW)	0
1	0 0 1 0 0 0 1 0	B B 7 B B B 7 B	7 (CYAN)	B (LT. YELLOW)	1
2	0 0 0 1 0 1 0 0	B B B C B C B B	C (GREEN)	B (LT. YELLOW)	2
3	0 0 0 0 1 0 0 0	B B B B E B B B	E (GRAY)	B (LT. YELLOW)	3
4	0 0 0 0 1 0 0 0	B B B B 8 B B B	8 (MED. RED)	B (LT. YELLOW)	4
5	0 0 0 0 1 0 0 0	B B B B 5 B B B	5 (LT. BLUE)	B (LT. YELLOW)	5
6	0 0 0 0 1 0 0 0	B B B B 6 B B B	6 (DK. RED)	B (LT. YELLOW)	6
7	0 0 0 0 1 0 0 0	B B B B D B B B	D (MAGENTA)	B (LT. YELLOW)	7

FIGURE 8 - GRAPHICS MODE 2 PATTERN COLOR DEFINITIONS

3.5 MULTICOLOR GRAPHICS MODE

The VDP is in multicolor graphics mode when mode bit M2 is 1 and mode bits M1 and M3 are 0. Multicolor mode provides an unrestricted 64H x 48V color square display. Each color square is a solid color and consists of a block of 4 x 4 pixels. Each 4 x 4 pixel block may be any one of the 16 possible colors, independent of all other pixel blocks.

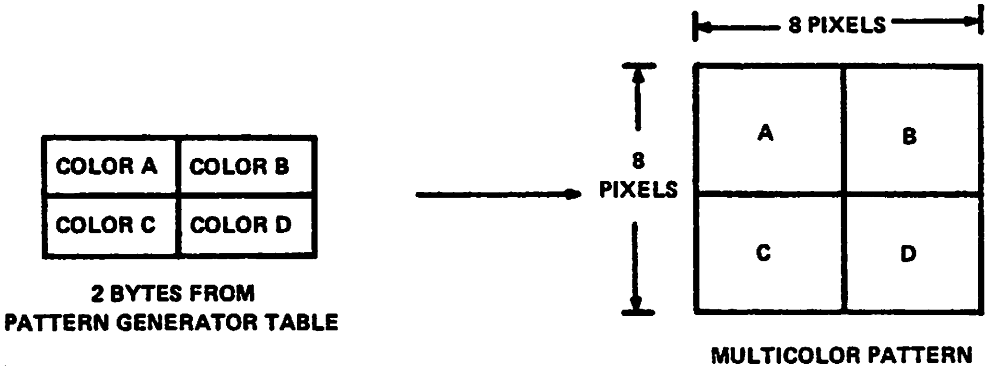


FIGURE 9 - MULTICOLOR CHARACTER CELL

The multicolor name table is the same as in graphics modes 1 and 2, consisting of 768 entries. However 'the pattern generator table is not used to define a tile pattern, as only one pattern is possible (a 4 x 4 pixel block), nor is the color table used to specify its color. Instead color information is derived directly from the pattern generator table, and the color table is unused.

As in graphics modes 1 and 2 each name table entry corresponds to one of the 768 8 x 8 pixel character cells. However these character cells are now divided into four 4 x 4 pixel blocks, as shown in Figure 9.

The name table, as usual, points to an 8 byte block in the pattern generator table, but only 2 bytes of this block are used to specify the screen image within the corresponding 8 x 8 pixel character cell. These 2 bytes specify the colors of the four 4 x 4 pixel blocks within the character cell. Refer to Figure 9. The high order nibble of the first byte defines the color of the upper left block, the low order nibble defines the color of the upper right block. Similarly the high and low order nibbles of the second byte define the colors of the lower left and lower right blocks respectively.

Which two bytes of the 8 byte block in the pattern generator table that are used to define the colors of the four blocks depends upon the the screen positions of the 4 x 4 pixel blocks. For the top two rows of 4 x 4 pixel blocks (defined by name table entries 0 - 31), the first two bytes of the 8 byte pattern generator table block are used. For the second two rows (defined by name table entries 32 - 63) the second two bytes are used, for the third two rows (defined by name table entries 64 - 95) the third two bytes are used, and for the fourth two rows (defined by name table entries 96 - 127) the last two bytes are used. This sequence repeats itself for the remainder of the screen.

For example, refer to Figures 10 and 11, if name table entry 0 contains the number N, pointing to the N th 8 byte block in the pattern generator table, then the 8 x 8 character cell corresponding to name table entry 0 will display the 8 x 8 multicolor pattern consisting of four 4 x 4 pixel blocks with the colors A, B, C and D. If name table entry 32 contains the same number N, pointing to the same 8 byte block in the pattern generator table, the 8 x 8 character cell corresponding to name table entry 32 will display the 8 x 8 multicolor pattern consisting of four 4 x 4 pixel blocks with the colors E, F, G and H instead. Likewise if name table entry 64 contained the same number N, the colors displayed in the multicolor pattern at the 8 x 8 pixel character cell 64 would be I, J, K and L; and if name table entry 96 contained this same number N, the colors displayed in the 8 x 8 pixel character cell 96 would be M, N, O and P.

The mapping of the VRAM contents to the screen image is simplified by using duplicate names in the name table, since the series of bytes used within the 8 byte pattern generator table segment specifies a 2 x 8 color square pattern on the screen as a straight forward translation from the 8 byte segment in the pattern generator table. The following method can be used to determine the value that should be placed in the name table entry for such a mapping.

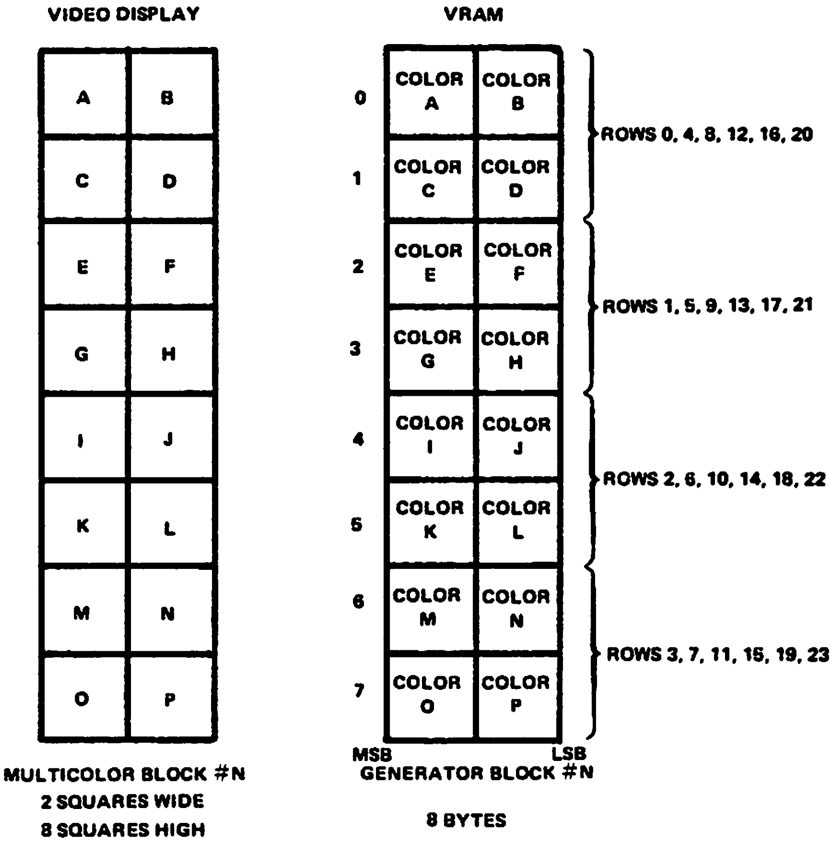


FIGURE 10 - MULTICOLOR PATTERN MAPPING

It is convenient to assign X and Y coordinates to the 32H x 24V character cell grid of Figure 4. The origin is in the upper left hand corner at character cell 0. Therefore (0, 0) would be character cell 0; (31, 0) would be character cell 31; (0, 1) would be character cell 32; and (31, 23) would be character cell 767. The name table entry N that corresponds to each (X, Y) coordinate is:

N = 32 * Y + X

The data P to be placed into this entry can be calculated by:

P = 32 * INT (Y / 4) + X

or in terms of N:

P = 32 * INT (INT (N/32) / 4) + N - 32 * INT (N/ 32)

Where INT means to take the integer portion only of the expression in parentheses.

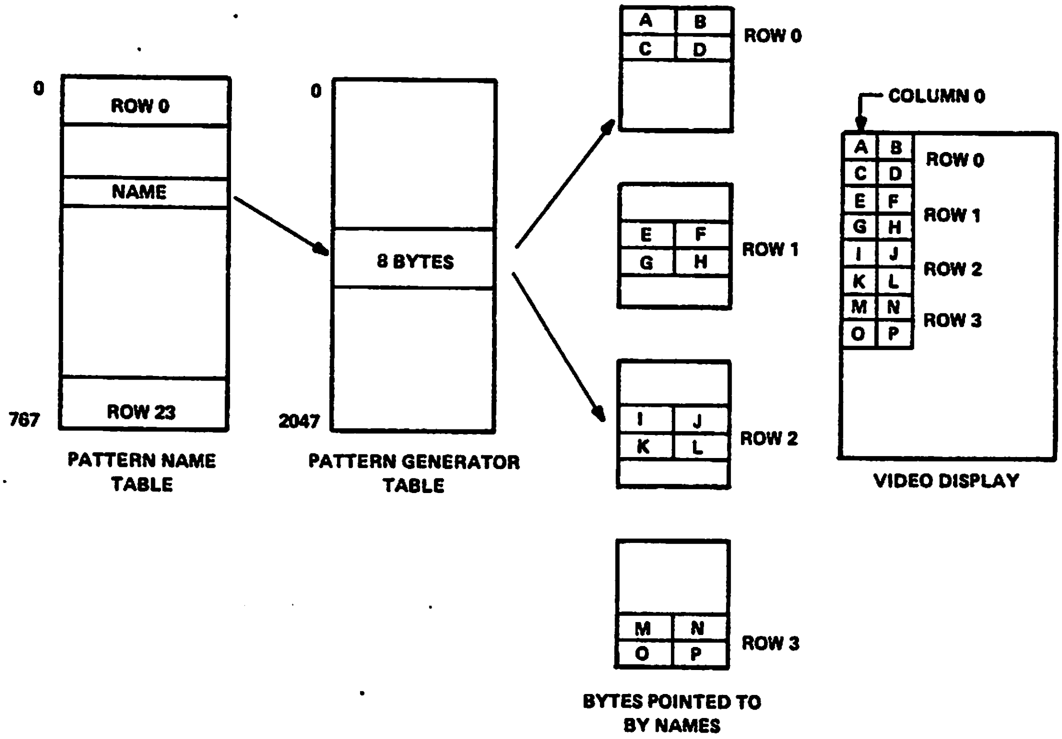


FIGURE 11 - MULTICOLOR MODE MAPPING

3.6 TEXT MODE

The VDP is in text mode when mode bit M1 is 1 and mode bits M2 and M3 are 0. In this mode the screen is divided into a grid of 40 text positions across and 24 down. See Figure 12. Each of the text positions contains a 6H x 8V pixel character cell. Only the name and pattern generator tables are used in text mode.

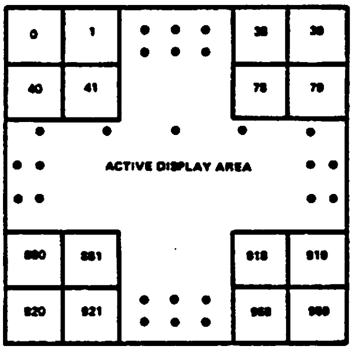


FIGURE 12 - TEXT MODE PATTERN PLANE

There can be up to 256 unique 6 x 8 pixel text patterns defined at any time. The name table maps the text pattern definitions into the 960 character cells on the the pattern plane. See Figure 13. Sprites are not available in text mode.

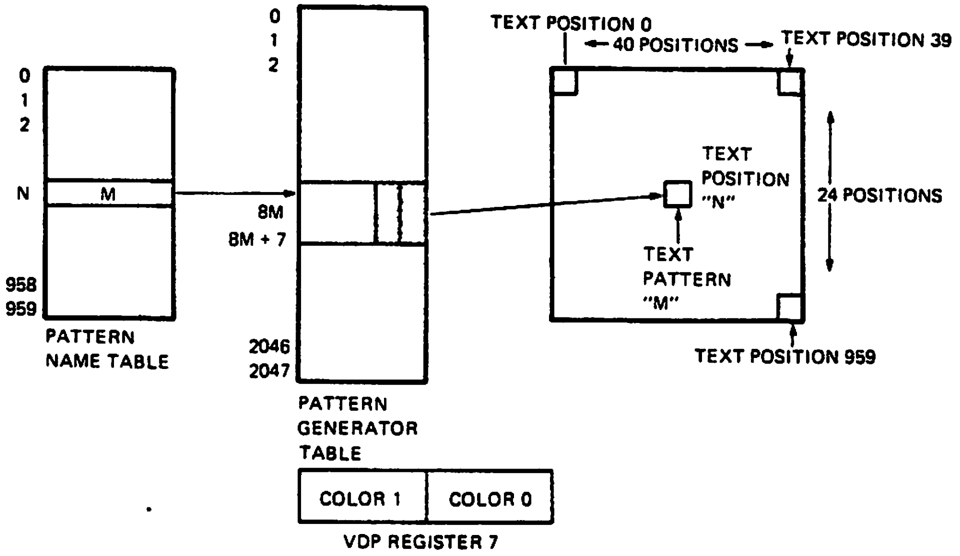


FIGURE 13 - TEXT MODE MAPPING

As with the graphics modes, the pattern generator table contains a library of text patterns that can be displayed in the text positions. It is 2 K bytes long and is arranged in 256 text patterns, each of which is 8 bytes long. Since each text position on the screen is only 6 pixels across, the least significant 2 bits of each text pattern are ignored, yielding 6 x 8 bits in each text pattern. Each 8 byte block defines a text pattern in which all the 1s in the text pattern take on one color when displayed on the screen, while all the 0s take on another color. These colors are chosen by loading VDP register 7 with the color 1 and color 0 values into the high and low order nibbles, respectively.

In short, the text mode is similar to graphics mode 1 with the following differences. The name table is 960 bytes long instead of 768, reflecting the grid of 40H x 24V text positions instead of 32H x 24V. The pattern generator table is identical, except the least 2 significant bits of each byte are unused to reflect the 6 x 8 character cell size. The color table is unused. Only two colors are available, the '0' color, which is the same as the backdrop color, and the '1' color. Both colors are determined by VDP register 7. Sprites are not available.

3.7 SPRITES

The video display can have up to 32 sprites on the highest priority video planes. The sprites are special animation patterns which provide smooth motion and multilevel pattern overlaying. The location of a sprite is defined by the top left-hand corner of the sprite pattern. The sprite can be easily moved pixel-by-pixel by redefining the sprite's X-Y position. This provides a simple but powerful method of quickly and smoothly moving these patterns. The sprites are not active in the text mode. The 32 sprite planes are fully transparent outside of the sprite itself.

The 32 sprites are numbered from 0 to 31. The sprites have relative priorities with sprite 0 having the highest priority and sprite 31 having the lowest. Sprite 0 is on the sprite plane closest to the viewer, while sprite 31 is on the sprite plane directly in front of the pattern plane. See Figure 3. When a sprite of higher priority passes over a sprite of lower priority, the overlapping pixels of the lower priority sprite seem to disappear behind the pixels of the higher priority sprite. All sprites have priority over the pattern plane, the backdrop plane and the external video plane.

Sprites comes in two sizes each with a choice of normal magnification (magnification 0) and double magnification (magnification 1). The size 0 sprites are 8 x 8 pixel patterns similar to tile patterns. Size 1 sprites are 16 x 16 pixel patterns and are actually composed of four 8 x 8 pixel patterns. When the magnification is set to 0 the sprites appear normal sized on the screen. When the magnification is set to 1, the sprites appear doubled in size on the screen, but the resolution remains the same. Therefore an 8 x 8 pixel sprite (size 0, magnification 0) becomes a 16 x 16 pixel sprite (size 0, magnification 1), but with only 8 x 8 pixel resolution. The 'pixels' actually become larger, equal to 4 regular pixels. Similarly a 16 x 16 sprite (size 1, magnification 0) becomes a 32 x 32 pixel sprite (size 1, magnification 1), but still with only 16 x 16 pixel resolution. The size and magnification of the sprites are determined by the SIZE and MAG bits of VDP register 1. See Section 2.1.2. See Table 3 for the relation between the SIZE and MAG bits and the sprite pattern. All 32 sprites must have the same size and magnification values.

SIZE	MAG	AREA	RESOLUTION	BYTES/PATTERN
0	0	8 × 8	single pixel	8
1	0	16 × 16	single pixel	32
0	1	16 × 16	2 × 2 pixels	8
1	1	32 × 32	2 × 2 pixels	32

TABLE 3 - SPRITE PATTERN FORMATS

The two tables in VRAM that define sprites are the sprite attribute table and the sprite pattern generator table. These tables are similar to their equivalents in the pattern plane realm in that the sprite attribute table specifies where the sprite goes on the screen and what color it is, while the sprite pattern generator table describes what the sprite pattern looks like. The sprite pattern generator table works identically to the graphics mode 1 pattern generator table. In fact the graphics mode 1 pattern generator table may also serve as the sprite pattern generator table. In other words, the tile patterns used in graphics mode 1 may also be used as sprites. Whereas the color table of graphics mode 1 determines the color of the tile patterns, the sprite attribute table determines the color of the sprite patterns.

The sprite pattern generator table contains up to 256 8 x 8 pixel sprite patterns. Each sprite pattern is defined by an 8 byte entry in the sprite pattern generator table. These 8 bytes define the sprite pattern in exactly the same way the graphics mode 1 pattern generator table defines the tile patterns. Sprite pattern 0 is the first entry in the sprite pattern generator table, sprite pattern 1 is the second entry and so on. All the 0 bits will be transparent. The 1 bits will be colored, but the color depends upon the color of the particular sprite, if any, that is using that sprite pattern. The sprite pattern generator table may be up to 256 x 8 = 2048 (2 K) bytes long and must start on a 2 K byte boundary. See Section 2.1.7.

The sprite attribute table determines the vertical position, the horizontal position, the sprite pattern, and the sprite color. The sprite attribute table consists of 32 4 byteentries. The first entry specifies the attributes of sprite 0, the highest priority sprite. The second entry specifies the attributes of sprite 1, the next highest priority sprite, and so forth. The sprite attribute table can be up to 4 x 32 = 128 bytes long. The sprite attribute table must begin on a 128 byte boundary, and there are 128 possible starting locations for this table. See Section 2.1.6. The meaning of the 4 bytes of each entry of the sprite attribute table are shown in Figure 14.

The first two bytes of each entry of the sprite attribute table determine the position of the sprite on the display. The first byte indicates the vertical distance of the sprite from the top of the active display area of the screen, in pixels. It is defined such that a value of -1 ($FF or 255) puts the sprite butted up at the top of the active display area of the screen, touching the backdrop border area. The second byte describes the horizontal displacement of the sprite from the left edge of the active display area. A value of 0 butts the sprite up against the left edge of the active display area, touching the backdrop border. Note that all measurements are taken from the upper left pixel of the sprite.

When the first two bytes of a sprite attribute table entry position a sprite so that it overlaps the backdrop border area, the part of the sprite that is within the active display area is displayed normally. The part of the sprite that overlaps the border is hidden from view by the border. This allows the animator to move a sprite into the display from behind the backdrop border area.

BYTE					
0	VERTICAL POSITION				
1	HORIZONTAL POSITION				
2	NAME				
3	EARLY CLOCK BIT	0	0	0	COLOR CODE

FIGURE 14 - SPRITE ATTRIBUTE TABLE ENTRY

The displacement in the vertical position is partially signed so that vertical displacement values from -31 ($E1) to 0 allow a sprite to bleed in from the top edge of the active display area of the screen. Vertical displacement values in the vicinity of 191 allow a sprite to bleed in from the bottom edge of the active display area. Similarly, horizontal displacement values in the vicinity of 255 allow a sprite to bleed in from the right side of the active display area. To allow sprites to bleed in from the left edge of the active display area, a special bit in the fourth byte of the sprite attribute table entry is used. This bit, called the early clock (EC) bit, when set to 0 does nothing, but when set to 1, the horizontal position of the sprite is shifted to the left by 32 pixels.

The third byte of each sprite attribute table entry determines which of the 256 patterns in the sprite pattern generator table is to be 'carried' by the sprite. This byte functions in much the same way as the bytes is the graphics mode 1 name table, which determine which of the 256 tile patterns will appear in the corresponding character cell locations.

The fourth byte of the sprite attribute table entry is used to determine the color of the sprite. The low order nibble determines which of the possible 16 colors the 1 bits of the pattern 'carried' by the sprite will be. The most significant bit is the early clock bit mentioned above.

There is a restriction on the number of sprites than can appear on a horizontal line. At most, only four sprites at a time can be active on any display line. If any of the 192 scan lines which makes up the active display area passes through five or more sprites, only the pixels of the four highest priority sprites will be seen. The pixels of all lower priority sprites will be transparent for that scan line. This is true even if the higher priority sprites are transparent on the scan line. Only those sprites that are active on a display line will cause the coincidence flag in the VDP status register to set. The status register provides the 5S flag bit and the number of the fifth sprite whenever five or more sprites are on a horizontal line.

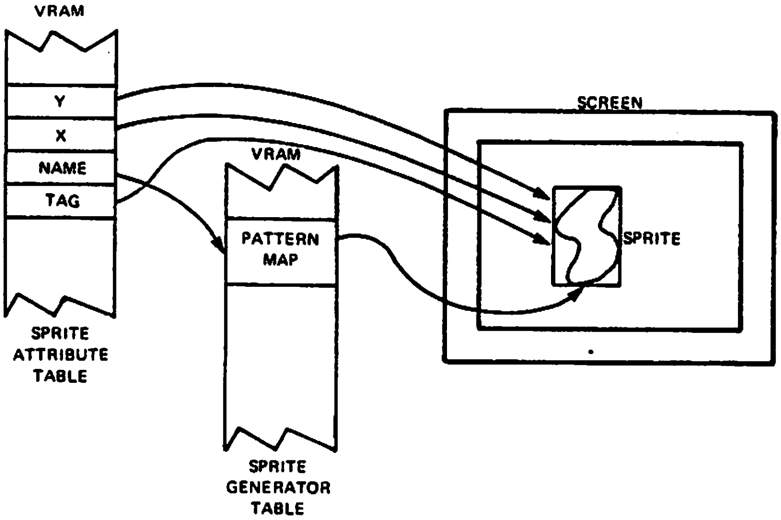


FIGURE 15 - SPRITE MAPPING

Whenever size 1 sprites, either 16 x 16 pixel or 32 x 32 pixel, are used the sprite pattern is determined by four consecutive 8 x 8 sprite patterns in the sprite pattern generator table. This means that there are only 256 / 4 = 64 different size 1 sprite patterns available, as each group of four 8 x 8 sprite patterns, starting from sprite pattern 0, now defines a single size 1 sprite pattern. The first of these four consecutive 8 x 8 patterns can not start just anywhere, but must be a pattern number which is an integer multiple of 4. Therefore sprite patterns 0 - 3 define one size 1 sprite pattern, sprite patterns 4 - 7 define another, sprite patterns 8 - 11 define a third, and so on.

Also, the value in the third byte of the sprite attribute table, which contains the sprite pattern number, must be an integer multiple of 4. 0, 4, 8, 12, 16, etc. are valid sprite pattern numbers for size 1 sprites. Any other numbers will be rounded down to the next lowest integer multiple of 4. Therefore the 'first' size one sprite pattern available is 0, the 'second' is 4, the 'third' is 8, etc.

The mapping of the 4 consecutive 8 x 8 sprite patterns to the size 1 sprite pattern are shown in Figure 16.

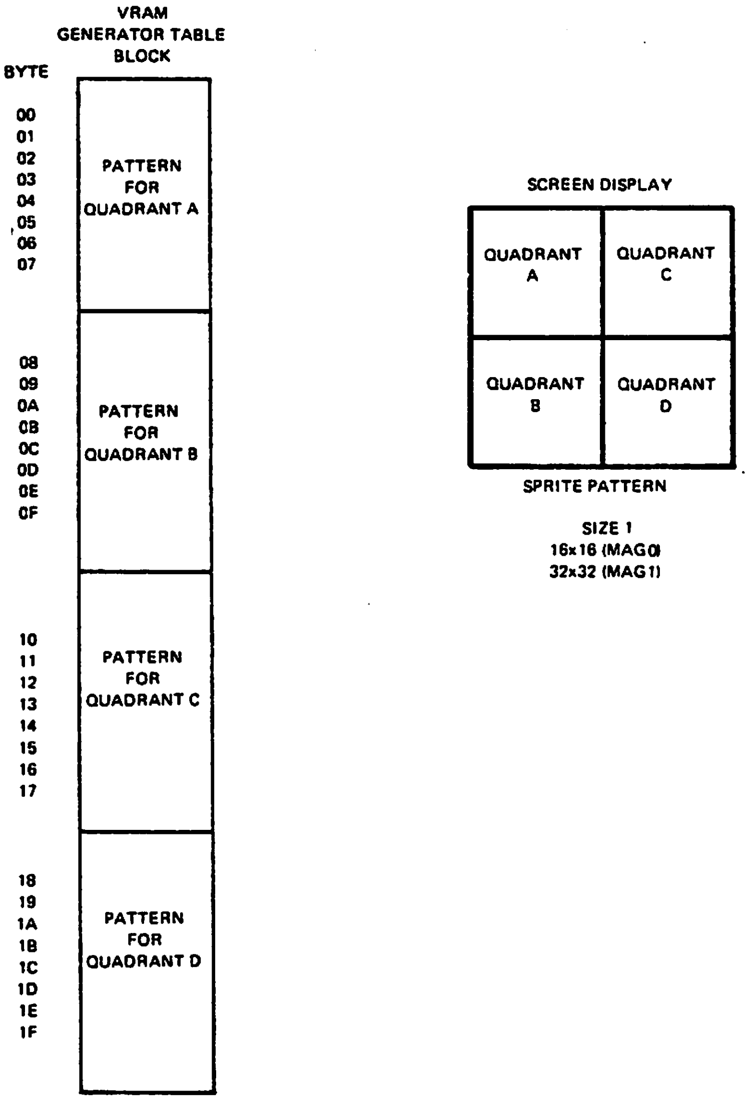


FIGURE 16 - SIZE 1 SPRITE MAPPING

Sprite processing is terminated if the VDP finds a value of 208 ($D0) in the vertical position field of any entry in the sprite attribute table. This permits the sprite attribute table to be shortened to the minimum size required; it also permits the user to blank out part or all of the sprites by simply changing one byte in VRAM. If the sprite terminator byte of 208 is not placed into the first unused sprite's vertical position byte in the sprite attribute table, unwanted garbage may appear on the screen. If no sprites are required then 208 must be placed in the very first byte of the sprite attribute table.

4.0 VDP I/O PORT ADDRESSES AND FUNCTIONS

To program the TMS 9918A Video Display Processor (VDP), data must be sent to the 8 control registers inside the VDP or to the 16K of on-board video RAM. The 8 control registers inside the VDP determine such things as which graphics mode the VDP is to display, at what locations in the video RAM the various graphics tables necessary for the chosen graphics mode are to be found, what the sprite size and magnification are to be, whether interrupts are to be enabled, what the backdrop color is to be, etc. The 16K of video RAM contain the information which determines what the graphics display will look like, just as Hires pages 1 and 2 of regular Apple graphics contain similar information. In addition the VDP contains a single status register which can be read to indicate such things as whether two sprites have collided, as well as other useful information. The data stored in the 16K of VRAM may also be read, if desired.

The 16K of video RAM is I/O mapped, that is it takes up none of the address space of the Apple's 6502 microprocessor. Therefore you can not read from or write to this 16K of RAM in the same manner as you would acces the RAM on the Apple's motherboard or on a 16K RAM card. All communication with this 16K of video·RAM must be done through the VDP. This process consists of first sending the address of the location you desire to access in the video RAM to one of the VDP I/O ports (either VREG or WREG) and then reading data from or writing data to another of the VDP I/O ports (either VRAM or WVRAM). All communication between the Apple's 6502 microprocessor and the video RAM must be done indirectly under the control of and via the VDP.

To send data either to the 8 VDP control registers or to the video RAM, or to read data from the VDP status register or from the video RAM, you must use the I/O ports provided on the Arcade Board for these purposes. Basically there are only two I/O ports (called VREG and VRAM) that are necessary to program the VDP. These two ports are both read-write (R/W) ports, that is you can both read data from or write data to these I/O ports. If you program the VDP in machine language, then all you will need to use are these two ports. However, we have provided two auxilliary write-only (W/O) ports called WREG and WVRAM. These ports can be used in place of VREG and VRAM for WRITING data, but since they are write-only ports, they can not be used for READING data. They are provided specifically to be used with the BASIC POKE instruction, so that you may program the VDP in either Integer or Applesoft BASIC. You can read data from both VREG and VRAM with a BASIC PEEK instruction, but you will get incorrect results if you try to write data to VREG or VRAM using a BASIC POKE instruction. Therefore the two auxilliary ports, WREG and WVRAM, are provided. BASIC programmers must use VREG and VRAM only for reading data (using the PEEK instruction), and the auxialliary ports, WREG and WVRAM, only for writing data (using the POKE instruction). Machine language programmers can use VREG and VRAM for both reading and writing data; they need not bother with WREG or WVRAM.

4.1 VREG: ADDRESSES AND FUNCTIONS

The VREG I/O port is used to communicate with the VDP registers. You may use this port to read the VDP status register or to write data to the 8 VDP control registers. The VDP status register is a read-only register (that is you can not write data to it), and the 8 VDP control registers are write-only registers (that is you can not read their contents).

VREG has two additional functions. As mentioned above, the 16K of video RAM can be accessed only via the VDP. In general, to access a given location in video RAM, you must first specify the address of the video RAM location that you wish to access. You must pass this address to the VDP through the VREG port. This is called 'setting-up the address'. Once the address has been set-up, you can read data from the video RAM or write data to the video RAM via the VRAM port.

The address of this port depends upon which slot the Arcade Board is inserted into. The address is given by:

In Hexadecimal:	VREG = $CON1
In Applesoft BASIC:	VREG = 49281 + 16 * S
In Integer BASIC:	VREG = 16 * S - 16255

N = slot number plus 8 (in hexadecimal) and S = slot number.

The address of the associated auxilliary port WREG is given by:

In Hexadecimal:	WREG = $CON3
In Applesoft BASIC:	WREG = 49283 + 16 * S
In Integer BASIC:	WREG = 16 * S - 16253

N = slot number plus 8 (in hexadecimal) and S = slot number.

REMEMBER: If you program the VDP in machine language, then you may use VREG for reading the VDP status register, writing data to the 8 VDP control registers, or for setting-up the address for a video RAM read or write operation. You do not need to use the WREG port at all. However, if you program the VDP in BASIC, you may use VREG only for READING the VDP status register, using a PEEK (VREG) instruction. To WRITE data to the 8 VDP control registers, or to set-up an address for a video RAM read or write operation, you must use the auxilliary port WREG. You can not POKE data to VREG. This is because the BASIC POKE instruction does weird things. Never try to POKE data to VREG, or to PEEK data from WREG. Instead POKE data to WREG, or PEEK data from VREG.

4.2 VRAM: ADDRESSES AND FUNCTIONS

The VRAM I/O port is used to read data from or write data to the video RAM. Once the address of the video RAM location you wish to access has been set-up (via the VREG port), then you would either read data from or write data to video RAM through the VRAM I/O port. You do not need to set-up the video RAM address prior to each read or write operation, if the location you wish to access follows sequentially after the last address accessed. For example, to write to the first 256 locations in video RAM (addresses 0 to 255), you would set-up the VDP for a write- to - video-RAM operation starting from address 0. Once this has been done, you may write data to address 0 in video RAM via the VRAM I/O port. To write data to addresses 1 to 255, merely send the data sequentially through the VRAM port. The data will automatically be placed in the next sequential address in video RAM without having to set-up the address each time. The VDP has an auto-incrementing address counter, which makes this possible.

The address of the VRAM I/O port depends upon which slot the Arcade Board is inserted into. The address is given by:

In Hexadecimal:	VRAM = $CONO
In Applesoft BASIC:	VRAM = 49280 + 16 * S
In Integer BASIC:	VRAM = 16 * S - 16256

N = slot number plus 8 (in hexadecimal) and S = slot number.

The address of the associated auxilliary port WVRAM is given by:

In Hexadecimal:	WVRAM = $CON2
In Applesoft BASIC:	WVRAM = 49282 + 16 * S
In Integer BASIC:	WVRAM = 16 * S - 16254

N = slot number plus 8 (in hexadecimal) and S = slot number.

REMEMBER: If you program the VDP in machine language, you may use the VRAM port for both reading data from or writing data to the video RAM. You need not ever use the auxilliary port WVRAM. However, if you program the VDP in BASIC, then you may use the VRAM port only for READNG data from the video RAM (using a PEEK instruction), but you may not use VRAM for WRITING data to video RAM with a POKE instruction. You must use the auxilliary port WVRAM for this purpose. Never try to POKE data to VRAM or PEEK data from WVRAM, but rather POKE data to WVRAM and PEEK data from VRAM.

4.3 READING THE VDP STATUS REGISTER

To read the contents of the VDP status register, merely execute a read of the VREG I/O port. In BASIC this would be something like STATUS = PEEK (VREG), and in machine language you would do something like LDA VREG.

NOTE: THERE ARE TWO VERY IMPORTANT REASONS TO READ THE VDP STATUS REGISTER, EVEN WHEN NO STATUS INFORMATION IS REQUIRED.

Since all operations involving writing to the VREG I/O port, (writing to the VDP registers and VRAM read / write address set up) involve two byte data transfers, the VDP must have some way of knowing which is the first byte and which is the second byte. Whenever the VDP's status register is read, the internal logic is reset so that the next byte written to VREG will be taken as the first byte of the two byte data transfer. If this is not done, the VDP may take the first byte as the second byte and vice versa. This is especially important in interrupt-driven environments. Whenever the status of the VDP's internal logic is in doubt, the status register should be read. The VDP'S internal logic is also reset upon power up and when the VDP is externally reset.

In a VDP interrupt-handling routine, one of the first things that must be done is to read the VDP's status register. This assures that the next byte written to VREG will be taken as the first byte of the two byte data transfer. If, in the main program, a two byte data transfer to VREG is interrupted before the second byte is transferred, the first byte sent by the interrupt routine will be treated as the second byte of the transfer by the VDP, unless the VDP status register is read, resetting the internal logic. Reading the VDP's status register also enables future VDP interrupts, as the VDP will not generate further interrupts until the last VDP generated interrupt is cleared, by reading the status register.

4.4 WRITING DATA TO THE VDP REGISTERS

Sending data to any one of the 8 VDP control registers consists of a two-byte transfer operation, in which the first byte sent is the actual data byte for the register, and the second byte is the selected register number (0 to 7) with the high bit set (add 128 to the register number). Both bytes of data must be sent to the VREG I/O port (or the WREG I/O port, if you use the BASIC POKE instruction). For example, if you wish to place a 15 into VDP register 2, you would go through the following sequence:

In machine language:

```
        LDA #15     ;load accumulator with data for reg 2
        STA VREG    ;send data to the VREG I/O port
        LDA #130    ;load accumulator with reg # plus 128
        STA VREG    ;send reg # plus 128 to VREG I/O port
```

In BASIC:

```
       POKE WREG,15 : REM  SEND DATA FOR REG 2 FIRST
       POKE WREG,130 : REM  THEN SEND REG # PLUS 128
```

4.5 READING DATA FROM THE VIDEO RAM

Reading data from the video RAM requires that you first specify the address within the 16K of on-board video RAM that you wish to read data from. The VDP contains an address counter which auto-increments upon each read or write video RAM operation. Therefore once the address you wish to read or write is set-up, you may do sequential reads without having to specify the address each time. If however, the next address you wish to read does not follow sequentially after the last address read, then you will have to set-up the address counter for the new address before you can read the data stored there.

Setting-up the VDP address counter is a two byte transfer operation. Since the video RAM has 16K of addresses, 14 bits are necessary to specify 16K of unique addresses. To set-up the VDP for a read of the video RAM, you have to specify a 14 bit address. Since the Apples's 6502 microprocessor is only an 8 bit processor, this requires sending the 14 bit address in two chunks. First you must send the low 8 bits of the address to the VREG I/O port and then the high 6 bits of the address to the same port. Make certain that the two most significant bits of the second byte are reset (=0). In machine language it is easy to split up the 14 bit address into a low 8 bits and a high 6 bits. In BASIC it requires a little math (see how below in the example). Once you have set-up the address to read, then all you need do is read data from the VRAM I/O port address. There is only one requirement; you must wait 8 microseconds before reading the first byte of data just after setting up the address. Thereafter you need wait only 3 microseconds after each data read. BASIC programmers need not worry about this delay.
As an example, suppose you wish to read the data at video RAM address ADDR. This is how you would set-up the VDP to read the data at that address:

In machine language:

```
LDA #ADDRL  ;get low 8 bits of ADDR
STA VREG    ;send to VREG I/O port
LDA #ADDRH  ;get high 6 bits of ADDR
STA VREG    ;send to VREG I/O port
NOP         ;2 microsecond delay
NOP         ;2 microsecond delay
NOP         ;2 microsecond delay
NOP         ;total of 8 microseconds delay
LDA VRAM    ;read data from ADDR
;do something with data and then
LDA VRAM    ;read next byte of data
;repeat as desired
```

In BASIC:

```
AH = INT (ADDR / 256) : REM  GET HIGH 6 BITS OF ADDR
AL = ADDR - 256 * AH : REM  GET LOW 8 BITS OF ADDR
POKE WREG, AL : REM  SEND LOW 8 BITS OF ADDR
POKE WREG, AH : REM  SEND HIGH 6 BITS OF ADDR
D1 = PEEK (VRAM) : REM  READ DATA FROM VRAM I/O PORT
D2 = PEEK (VRAM) : REM  READ NEXT BYTE OF DATA
REM  REPEAT AS DESIRED
```

Note: In the machine language example ADDRL and ADDRH are used to indicate the low and high bytes respecitively of the 16 bit quantity ADDR (the 2 most significant bits should always be 0 so in effect ADDR is only a 14 bit quantity). Your assembler may have a different nomenclature for this operation. In the BASIC example the low 8 bits and the high 6 bits must first be calculated. This is done by dividing the address ADDR by 256 and taking the integer portion only to give the high 6 bits of the address. To get the low 8 bits the high 6 bits multiplied by 256 is subtracted from the address ADDR. Remember if you wish to read an address which does not follow sequentially after the last address read, then you will have to set-up the VDP's address counter for the new address, before you can read the data there.

4.6 WRITING DATA TO THE VIDEO RAM

Writing data to the 16K of on-board video RAM is very similar to reading the data. You must first set-up the VDP's address counter with the address of the first byte of data you wish to write, and then you may write as many sequential bytes of data as you desire. There is a slight difference in setting-up the address counter as compared with the read data operation. When you send the high 6 bits of the address you must also set bit 6, that is you must add 64 to the quantity. This bit acts as a code to the VDP telling it that it is a WRITE set-up as opposed to a READ set-up (in which case bit 6 is reset, i.e. is set equal to 0). Once the address is set-up you merely write as many bytes of data to the VRAM I/O port as you desire. As an example, suppose you wish to send several bytes of data to the video RAM starting from address ADDR:

In machine language:

```
LDA #ADDRL   ;get low 8 bits of address ADDR
STA VREG     ;send to VREG I/O port
LDA #ADDRH + 64  ;get high 6 bits of ADDR plus 64
STA VREG     ;send to VREG I/O port
LDA #DATA1   ;get first byte of data to send
STA VRAM     ;send to video RAM
LDA #DATA2   ;get second byte of data to send
STA VRAM     ;send to video RAM
;repeat as desired
```

In BASIC:

```
AH = INT (ADDR / 256) : REM  GET HIGH 6 BITS OF ADDR
AL = ADDR - 256 * AH : REM  GET LOW 8 BITS OF ADDR
POKE WREG,AL : REM  SEND LOW 8 BITS OF ADDR
POKE WREG,AH + 64 : REM  SEND HIGH 6 BITS OF ADDR + 64
```

```
        POKE WVRAM,D1 : REM  SEND FIRST BYTE OF DATA
        POKE WVRAM,D2 : REM  SEND SECOND BYTE OF DATA
        REM 'REPEAT AS DESIRED
```

Note: See the note above under the example for 'READING DATA FROM THE VIDEO RAM' for explanations.

In all write operations involving the VREG I/O port, two bytes of data are transfered. The two most significant bits (6 and 7) of the second byte act as a code to distinguish between the three possible operations: writing data to one of the control registers, a read video RAM address set-up, or a write to video RAM address set-up. Bit 7 when set indicates a write to a control register. When bit 7 is reset and bit 6 is reset, this indicates a read video RAM address set-up. When bit 7 is reset and bit 6 is set, then this indicates a write to video RAM address set-up.4.1 VDP Register Ports

5.0 VRAM ALLOCATION

The various VRAM tables of the four display modes of the VDP can be allocated in literally hundreds of different combinations of base addresses and degrees of overlap. The particular application for which the VDP is to be used will often determine how VRAM is to be allocated. For instance if 40 column text is required, then the text mode is needed. If sprites are needed, then only the three graphics modes can be used. If more than 256 unique tile patterns, or more colors per tile pattern are required, then graphics mode 2 is necessay. Even within each display mode there are many choices left on how the VRAM will be allocated to the various tables. In general the most efficient allocation is the one of choice, because this allows more room for other tables, even of differnt graphics modes, which can be swicthed to merely by changing a few of the VDP registers, and thus changing the whole screen in a flash. The most efficient allocation may involve overlap of tables if not all the capabilities of any given table are to be exploited. For example if only 128 different sprite patterns are needed, then there is no need for a full 2 K sprite pattern generator table, only 1 K is necessary. The remaining 1 K area can be used for other tables. Below are are just a few examples of VRAM allocation in the various modes. With the proper VRAM allocation, many spectacular effects are possible. The examples given below are only meant to show some of the possibilities of the VDP graphics. Many more combinations and uses are possible and left for the imagination.

5.1 GRAPHICS MODE 1 VRAM ALLOCATION

There are 3 pattern plane tables required in graphics mode 1 and 2 sprite tables.

There are many possibilities. For instance it is possible to have one 2 K long pattern generator table and one 2 K long sprite pattern generator table. This leaves room for 12 different name tables, each with two possible color tables and one sprite attribute table. There could then be 12 different pages of graphics mode 1 screens, each corresponding to one of the 12 different sets of name tables, color tables and sprite attribute tables. Any one of these 12 different screens can be selected by merely changing a few of the VDP registers. Of course, each of these screens will have to share the same 256 tile patterns and 256 sprite patterns. If this is a problem, a few extra name tables can be exchanged for more pattern generator or sprite pattern generator tables. It might even be possible to use the same table for both the tile pattern generator table and the sprite pattern generator table, saving a big chunk of VRAM.

It is also possible to have four completely independent screens of graphics mode 1 displays. Each screen has its own set of the five VRAM tables. This involves some small sacrifice. 1 K of the 2 K byte long sprite pattern generator table is overlayed by the name table, the color table, and the sprite attribute table. There is still room for 136 different size 0 sprite patterns, or 34 different size 1 sprite patterns. There is room in each of these screens for the full 256 different tile patterns. The Amparcade comands & GM1A, & GM1B, & GM1C, and & GM1D select one of these four pages of graphics mode 1. The memory allocation of each of these commands is shown in Table 4.

COMMAND	NT	CT	PGT	SAT	SPGT
& GM1A	1	30	1	14	0
& GM1B	5	94	3	46	2
& GM1C	9	158	5	78	4
& GM1D	13	222	7	110	6

TABLE 4 - VRAM ALLOCATION OF AMPARCADE & GM1 COMMANDS

5.2 GRAPHICS MODE 2 VRAM ALLOCATION

The 3 pattern plane tables and 2 sprite tables of graphics mode 2 require a considerable amount of memory, at least 12 K if the mode is to be utilized to it fullest capacity. There are not too many possibilities. First of all both the pattern generator table and the color table are 6 K bytes long, and must begin on an 8 K boundary. This leaves only two useful choices: either the pattern generator table begins at $0000 and the color table begins at $2000, or the pattern generator table begins at $2000 and the color table begins at $0000. This then limits the possible choices for the name table to either name tables 6, 7, 14 or 15. The sprite pattern generator table choices are 3 or 7. There is still lots of possible places to put the sprite attribute table. In fact, there could be several different sprite attribute tables. Changing VDP register 5, which is the sprite attribute table base pointer, will instantly change the status of the sprites on the screen.

Since the first 2 K blocks of both the pattern generator table and the color table define the top third of the screen, the second 2 K blocks define the middle third of the screen, and the last 2 K blocks define the bottom third of the screen, neither of these two tables can be simply cut into two contiguous halves, using only one half and leaving the other half free for overlaying other tables. At least some of the bytes in each 2 K block must be used for both the pattern generator table and the color table. However, not all of each 2 K block needs to be used. For example, if only 128 unique tile patterns are required in each third of the screen, then 1 K from each 2 K block, or 6 K in all, is freed for overlaying. It is even possible to have two contiguous 2 K blocks freed, with the remaining 2k non-contiguous. This can be done if, for example, the top third of the screen uses only patterns 0 to 127 and the middle third of the screen uses patterns 384 to 511, then the 2 K normally used by patterns 128 to 383 is free. A graphics mode 1 pattern generator table can be placed here, or another sprite pattern generator table, or even 2 more name tables, allowing at least 3 pages of graphics mode 2 screens. Of course there are still the other two 1 K blocks freed from the color and pattern generator tables to be used for a multitude of possible purposes.

The Amparcade command allocates VRAM according to the following table.

COMMAND	NT	CT	PGT	SAT	SPGT
& GM2	6	255	3	54	7

TABLE 5 - VRAM ALLOCATION OF AMPARCADE COMMAND & GM2

5.3 MULTICOLOR GRAPHICS MODE VRAM ALLOCATION

The multicolor graphics mode does not use color tables. In addition only 3 / 4 of the pattern generator table is used, when the name table is set up as specified in section 3.5. This leaves the last 1 / 2 K of the pattern generator free.

There is room in VRAM for four totally independent screens of multicolor mode graphics, each screen with its own set of name, pattern generator, sprite attribute and sprite pattern generator tables. The price to pay for this is that there is room for only 160 sprite patterns in each page's sprite pattern generator table. The Amparcade commands & GM3A, & GM3B, &GM3C and & GM3D allocate VRAM to one of these pages. The allocation of VRAM by these commands is shown in Table 6.

COMMAND	NT	CT	PGT	SAT	SPGT
& GM3A	1	63	1	28	0
& GM3B	5	127	3	60	2
& GM3C	9	191	5	92	4
& GM3D	13	255	7	124	6

TABLE 7 - VRAM ALLOCATION OF AMPARCADE COMMAND & GM3

5.4 TEXT MODE VRAM ALLOCATION

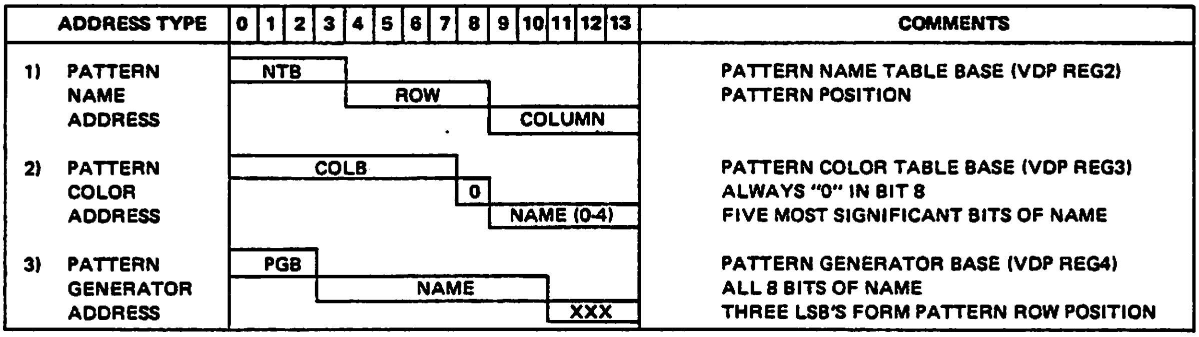
Most text mode applications require a single pattern generator table of up to 2 K bytes in length. Since sprites are not available in this mode, these tables do not need to be allocated. There is then room for 14 name tables, so there can be 14 pages of text held in VRAM at any given time.

6.0 VRAM ADDRESS DERIVATION

This section contains information on how to derive the 14 bit VRAM addresses for the entries, or bytes of each entry, of each of the various tables used in each display mode. The base addresses of the various tables are calculated from the formulas given in sections 2.1.3 to 2.1.7. The VRAM address of any given entry N is calculated by multiplying N by the number of bytes per entry and adding the product to the table's base address.

In the tables below, the number of the bits is reversed from normal convention. The highest numbered bits are actually the lowest order bits of the 14 bit address, and the lowest numbered bits are the highest order bits. Bit 0 is the highest order bits, normally called bit 13 in standard convention.

GRAPHICS I MODE ADDRESS LOCATION



ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
1) PATTERN	NTB														PATTERN NAME TABLE BASE (VDP REG2)
NAME					ROW										PATTERN POSITION
ADDRESS										COLUMN					
2) PATTERN	COLB														PATTERN COLOR TABLE BASE (VDP REG3)
COLOR									0						ALWAYS "0" IN BIT 8
ADDRESS										NAME (0-4)					FIVE MOST SIGNIFICANT BITS OF NAME
3) PATTERN	PGB														PATTERN GENERATOR BASE (VDP REG4)
GENERATOR				NAME											ALL 8 BITS OF NAME
ADDRESS												XXX			THREE LSB'S FORM PATTERN ROW POSITION

TABLE 7 - VRAM ADDRESS DERIVATION

GRAPHICS II MODE ADDRESS LOCATION

<table>
<tr><th>ADDRESS TYPE</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>COMMENTS</th></tr>
<tr><td rowspan="3">1) PATTERN
NAME
ADDRESS</td><td colspan="4">NTB</td><td colspan="10"></td><td>PATTERN NAME TABLE BASE (VDP REG2)</td></tr>
<tr><td colspan="4"></td><td colspan="5">ROW</td><td colspan="5"></td><td>PATTERN POSITION ROW</td></tr>
<tr><td colspan="9"></td><td colspan="5">COLUMN</td><td>PATTERN POSITION COLUMN</td></tr>
<tr><td rowspan="4">2) PATTERN
COLOR
ADDRESS</td><td></td><td colspan="13"></td><td>PATTERN COLOR TABLE BASE MSB (VDP REG3)</td></tr>
<tr><td></td><td colspan="2">XX</td><td colspan="11"></td><td>TWO MSB FROM VERTICAL COUNTER</td></tr>
<tr><td colspan="3"></td><td colspan="8">NAME</td><td colspan="3"></td><td>ALL 8 BITS OF NAME</td></tr>
<tr><td colspan="11"></td><td colspan="3">XXX</td><td>COLOR TABLE BYTE/LINE</td></tr>
<tr><td rowspan="4">3) PATTERN
GENERATOR
ADDRESS</td><td></td><td colspan="13"></td><td>PATTERN NAME TABLE BASE MSB (VDP REG4)</td></tr>
<tr><td></td><td colspan="2">XX</td><td colspan="11"></td><td>TWO MSB FROM VERTICAL COUNTER</td></tr>
<tr><td colspan="3"></td><td colspan="8">NAME</td><td colspan="3"></td><td>ALL 8 BITS OF NAME</td></tr>
<tr><td colspan="11"></td><td colspan="3">XXX</td><td>PATTERN GENERATOR BYTE/LINE NUMBER</td></tr>
</table>

TEXT MODE ADDRESS LOCATION

<table>
<tr><th>ADDRESS TYPE</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>COMMENTS</th></tr>
<tr><td rowspan="2">TEXT MODE
NAME ADDRESS</td><td colspan="4">NTB</td><td colspan="10"></td><td>PATTERN NAME TABLE BASE (VDP REG2)</td></tr>
<tr><td colspan="4"></td><td colspan="10">TEXT POSITION</td><td>EQUAL (TEXT POSITION ROW # TIMES 40) PLUS (TEXT POSITION COLUMN NUMBER)</td></tr>
<tr><td rowspan="3">TEXT MODE
PATTERN
ADDRESS</td><td colspan="3">PGB</td><td colspan="11"></td><td>PATTERN GENERATOR BASE (VDP REG4)</td></tr>
<tr><td colspan="3"></td><td colspan="8">NAME</td><td colspan="3"></td><td>NAME</td></tr>
<tr><td colspan="11"></td><td colspan="3">XXX</td><td>BYTE/LINE NUMBER</td></tr>
</table>

SPRITE ADDRESS LOCATION

<table>
<tr><th>ADDRESS TYPE</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>COMMENTS</th></tr>
<tr><td rowspan="3">SPRITE
ATTRIBUTE
ADDRESS</td><td colspan="7">SAB</td><td colspan="7"></td><td>SPRITE ATTRIBUTE TABLE BASE (VDP REG5)</td></tr>
<tr><td colspan="7"></td><td colspan="5">SPRITE</td><td colspan="2"></td><td>SPRITE NUMBER</td></tr>
<tr><td colspan="12"></td><td colspan="2">XX</td><td>ATTRIBUTE NUMBER:
00 FOR VERTICAL POSITION
01 FOR HORIZONTAL POSITION
10 FOR NAME
11 FOR TAG (EARLY CLOCK AND COLOR)</td></tr>
<tr><td rowspan="3">SIZE = 0
SPRITE PATTERN
GENERATOR</td><td colspan="3">SPGB</td><td colspan="11"></td><td>SPRITE PATTERN GENERATOR BASE (VDP REG4)</td></tr>
<tr><td colspan="3"></td><td colspan="8">NAME</td><td colspan="3"></td><td>NAME ATTRIBUTE OF SPRITE</td></tr>
<tr><td colspan="11"></td><td colspan="3">XXX</td><td>THREE LSB'S GIVE BYTE/LINE NUMBER</td></tr>
<tr><td rowspan="3">SIZE = 1
SPRITE PATTERN
GENERATOR</td><td colspan="4">SPGB</td><td colspan="10"></td><td>SPRITE PATTERN GENERATOR BASE (VDP REG4)</td></tr>
<tr><td colspan="4"></td><td colspan="5">NAME (0-5)</td><td colspan="5"></td><td>SIX MSB OF NAME</td></tr>
<tr><td colspan="9"></td><td colspan="5">XXXXX</td><td>SIZE = 1 SPRITE BYTE NUMBER (SEE FIGURE 4-4)</td></tr>
</table>

MULTICOLOR ADDRESS LOCATION

<table>
<tr><th>ADDRESS TYPE</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>COMMENTS</th></tr>
<tr><td rowspan="3">4) MULTICOLOR
NAME
ADDRESS</td><td colspan="4">NTB</td><td colspan="10"></td><td>NAME TABLE BASE (VDP REG2)</td></tr>
<tr><td colspan="4"></td><td colspan="6">ROW</td><td colspan="4"></td><td>PATTERN POSITION ROW</td></tr>
<tr><td colspan="10"></td><td colspan="4">COLUMN</td><td>PATTERN POSITION COLUMN</td></tr>
<tr><td rowspan="3">5) MULTICOLOR
COLOR
GENERATOR
ADDRESS</td><td colspan="3">PGB</td><td colspan="11"></td><td>PATTERN GENERATOR BASE (VDP REG4)</td></tr>
<tr><td colspan="3"></td><td colspan="7">NAME</td><td colspan="4"></td><td>NAME FROM NAME FETCH</td></tr>
<tr><td colspan="10"></td><td colspan="4">XXX</td><td>THREE LSB'S FORM BYTE/SQUARE ROW</td></tr>
</table>

TABLE 7 - VRAM ADDRESS DERIVATION (CONTINUED)

7.0 VDP INTERRUPTS

Whenever bit 5 in VDP register 1, the interrupt enable IE bit, is set to 1 and the F bit of the VDP status register is 1, the VDP INT pin becomes active and generates an interrupt at the end of the last line of the active display area, that is, at the beginning of the vertical retrace period. The INT pin is tied to the 6502 IRQ line, and thus the interrupt is maskable. The interrupt occurs about every 1 / 60 th of a second. The interrupt is cleared when the status register is read. For interrupt driven routines, the status register must be read before the next interrupt can be generated.

The interrupt is useful for synchronized page flipping or writing to VRAM during the vertical retrace period. This prevents unsightly screen glithes. It may also be used as a general purpose timer for games, music, interrupt driven routines, etc.

8.0 EXTERNAL VIDEO AND DUAL VDPS

The lowest priority plane of the VDP's multiplanar graphics structure is the external video plane. When the EV bit in VDP register 0 is enabled, the external video signal forms this plane's image. The other VDP planes can be superimposed upon this external video plane. This requires special hardware, and is not normally used. There are two exceptions that do not require special hardware. (1) If the backdrop color is set to transparent, the EV bit in VDP register 0 is set to 1, and the BLANK bit in VDP register 1 is reset to 0, then an external video signal may pass through the VDP unchanged (except diminished in intensity a little). (2) If the external video signal is from another VDP and if both VDPs are run off the same crystal, then all 35 of the second VDP's video planes become the external video plane of the first VDP. In this way 64 sprite planes are available, and two pattern planes, with on pattern plane sandwiched between two sets of 32 sprite planes. The Arcade Board allows for both of these possibilities, and with some external hardware, any video signal may also be used to form the external video plane image. Note: in all cases, in order for the external video plane to be visible the backdrop color plane and at least part of the pattern plane must be transparent.

9.0 TILE AND SPRITE PATTERN CREATION

The following pages give some examples of tile and sprite pattern creation. They require a knowledge of binary and hexadecimal numbers.

A Step-by-Step Approach to Create Patterns and Sprites

PATTERNS

1. Use an 8 × 8 pattern similar to that in Figure A. Each small square represents one pixel on the screen.

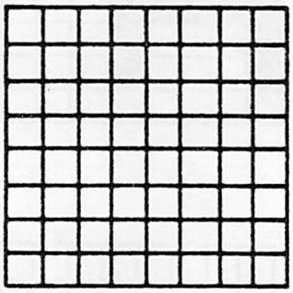


FIGURE A

2. Fill in the blocks to create your text character or graphics pattern. Examples of the letter A and an ARROW are shown in Figures B and C.

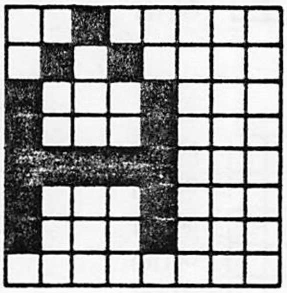


FIGURE B

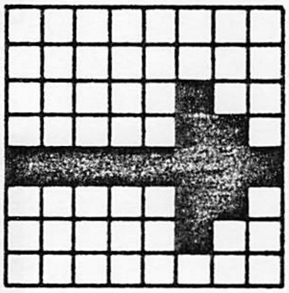
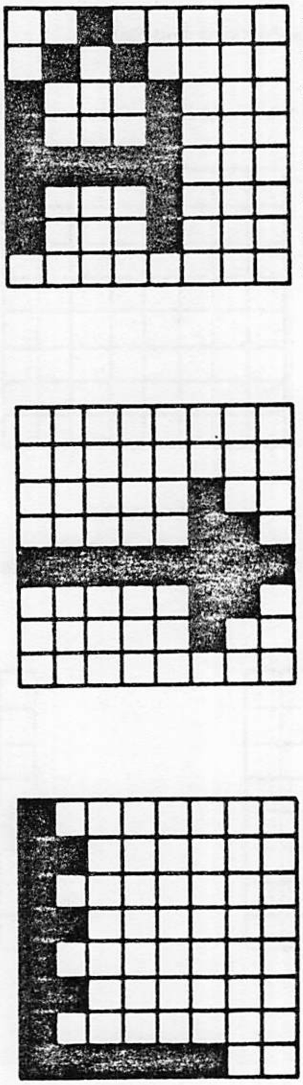


FIGURE C

NOTE

If these patterns are to be used in the Text mode, (40 patterns per line), the pattern should be inside a left-justified 6 × 8 block like the A shown in Figure B. If all of the Text patterns are inside this 6 × 8 block, they can be used for Text and Graphics 1 and 2 modes.

3. Assign 1s to the filled-in areas and 0s to the blanks. Then convert the 1s and 0s to their hexadecimal equivalents, as shown in Figure D.



= 00100000 = $20_{(16)}$

= 01010000 = $50_{(16)}$

= 10001000 = $88_{(16)}$

= 10001000 = $88_{(16)}$

= 11111000 = $F8_{(16)}$

= 10001000 = $88_{(16)}$

= 10001000 = $88_{(16)}$

= 00000000 = $00_{(16)}$

= $00_{(16)}$

= $00_{(16)}$

= $04_{(16)}$

= $06_{(16)}$

= $FF_{(16)}$

= $06_{(16)}$

= $04_{(16)}$

= $00_{(16)}$

= $80_{(16)}$

= $C0_{(16)}$

= $80_{(16)}$

= $C0_{(16)}$

= $80_{(16)}$

= $C0_{(16)}$

= $80_{(16)}$

= $FC_{(16)}$

FIGURE D

4. Now place the eight bytes defining the pattern into the Pattern Generator Table. Assume the Pattern Generator Table sub-block is located at 800_{16} and the arrow pattern is to be named 00_{16}. Then place the eight pattern bytes as follows:

800		PATTERN NAME 00
801	00	
802	04	
803	06	
804	FF	
805	06	
806	04	
807	00	
808		PATTERN NAME 01
809		
80A		
80B		
80C		
80D		
80E		
80F		
810		
900	00	PATTERN NAME 20
901	00	
902	00	
903	00	
904	00	
905	00	
906	00	
907	00	
908		
A08	20	PATTERN NAME 41
A09	50	
A0A	88	
A0B	88	
A0C	F8	
A0D	88	
A0E	88	

NOTE

When using text in your applications, you can place the eight bytes of the text character in its ASCII number location.

Example: ASCII SPACE = 20_{16}
? = $3F_{16}$
A = 41_{16}
B = 42_{16}
C = 43_{16}
Etc.

This simplifies writing text to the screen. Simply write the ASCII name directly to the Pattern Name Table.

A space character is shown in Pattern Generator Table position 20, and A is shown in pattern name 41.

SPRITES

1. Determine whether to use 8 × 8 or 16 × 16 sprite patterns. Then use the appropriate work pattern, as shown in Figures E and F.

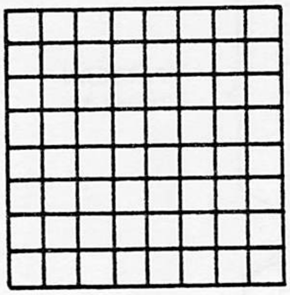


FIGURE E

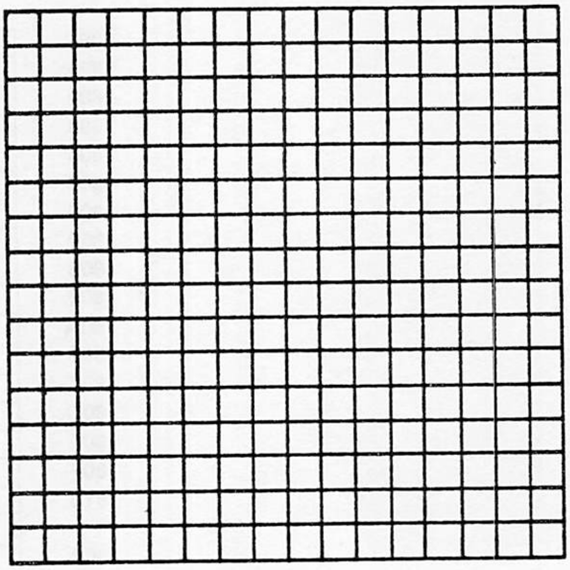


FIGURE F

2. Fill in the blocks to create your sprite pattern. Examples are shown in Figures G and H.

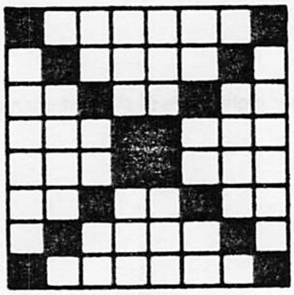


FIGURE G

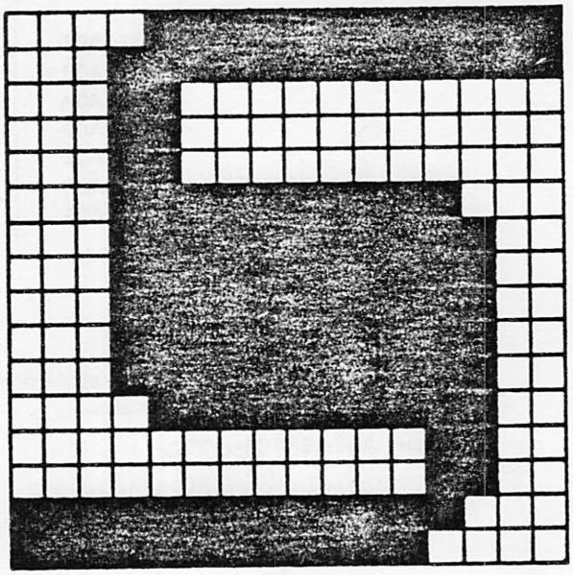


FIGURE H

3. Next encode the sprite patterns as in the Pattern Section. The 8 × 8 sprite encodes exactly as the 8 × 8 pattern, but the 16 x 16 sprite encodes as shown in Figure J.

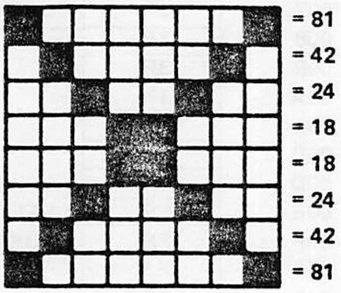


FIGURE I

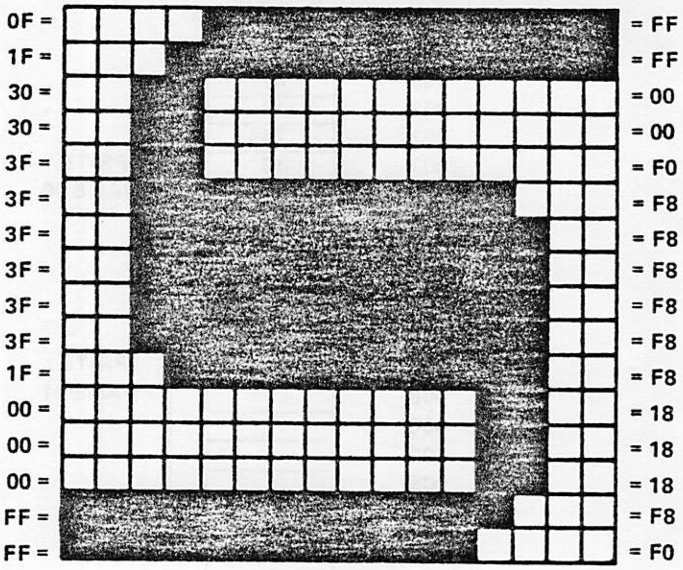
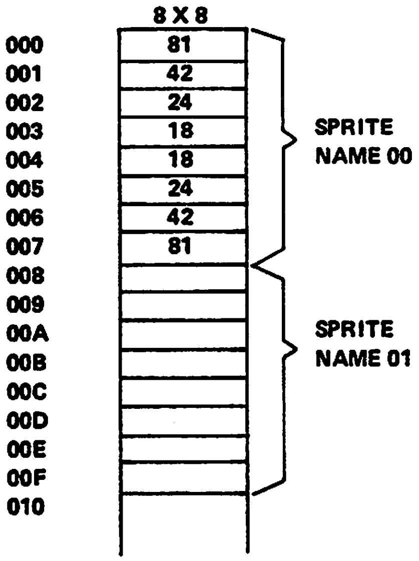


FIGURE J

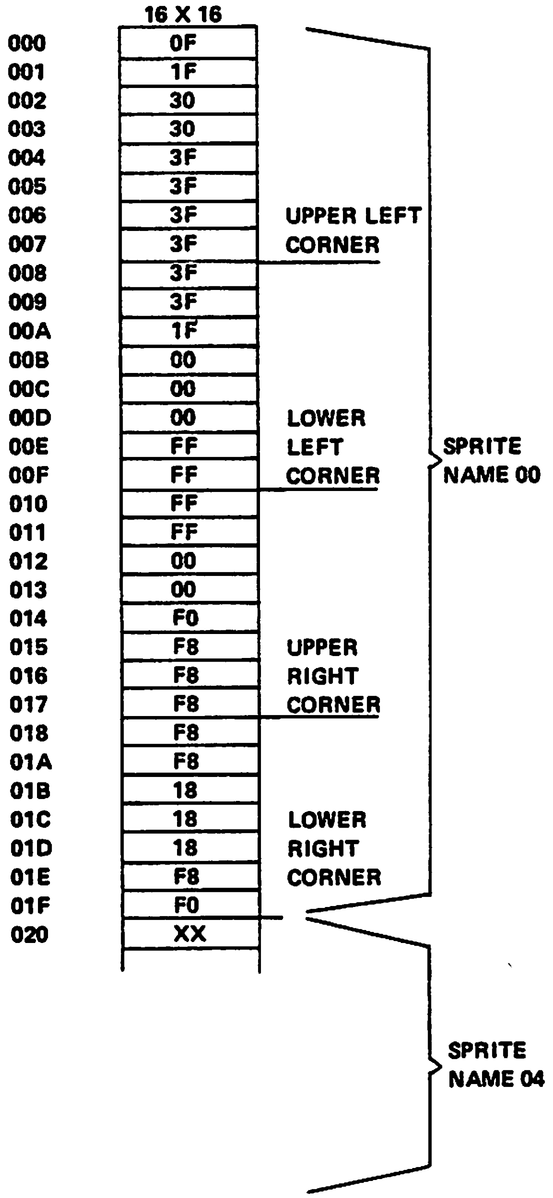
Break the 16 × 16 block pattern into four 8 × 8 patterns. Next, encode the 8 × 8 patterns starting in the upper left corner, then do the lower left, upper right, and lower right.

4. Place the 8 bytes for 8 × 8 sprites or 32 bytes for 16 × 16 sprites in the Sprite Generator Table. Assuming the sprite generator table is located at location 0000, Figures K and L show how the tables should look for 8 × 8 and 16 × 16 sprites.



	8 X 8
000	81
001	42
002	24
003	18
004	18
005	24
006	42
007	81
008	
009	
00A	
00B	
00C	
00D	
00E	
00F	
010	

FIGURE K



	16 X 16
000	0F
001	1F
002	30
003	30
004	3F
005	3F
006	3F
007	3F
008	3F
009	3F
00A	1F
00B	00
00C	00
00D	00
00E	FF
00F	FF
010	FF
011	FF
012	00
013	00
014	F0
015	F8
016	F8
017	F8
018	F8
01A	F8
01B	18
01C	18
01D	18
01E	F8
01F	F0
020	XX

FIGURE L

16 × 16 sprite patterns start in the table with the byte from the upper left-hand corner. Then start with the upper right, going toward the lower right.

VDP REFERENCES:

Electronics, November 20, 1980, pages 123-126: "Video Display Processor Simulates Three Dimensions", Karl Guttag and John Hayn

IEEE COMPCON 80, Proceedings of Distributed Computing, Twenty First IEEE Computer Society International Conference, Washington D.C., U.S.A., 23-25 September 1980, pages 219-223: "The TMS 9918 Video Display Processor: A Brief Overview", Karl Guttag

IEEE Transactions on Consumer Electronics, Volume CE-27, February 1981, pages 27-33: "Video Display Processor", Karl Guttag and Peter Macourek

BYTE, August 1982, pages 57-80: "High-Resolution Sprite-Oriented Color Graphics", Steve Ciarcia

80 Micro, May 1983, pages 90-102, "The 80 Goes Color, Part I", James Cole

80 Micro, June 1983, pages 116-130, "The 80 Goes Color, Part II", James Cole

Texas Instruments Inc., copyright 1982: "Texas Instruments 9900, TMS 9918A/TMS 9928A/TMS 9929A Video Display Processors Data Manual"

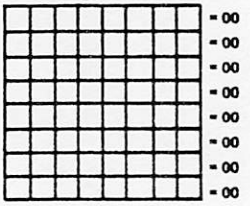
APPENDIX A

ASCII CHARACTER SET

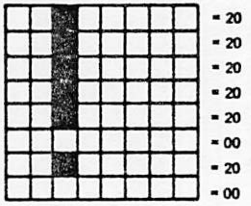
Software programs apply to all three VDPs (TMS9918A/9928A/9929A).

This appendix contains the diagrams and software listing of an upper and lower case ASCII character set. The character matrix is 5 X 7 in the 8 X 8 pixel block. These characters are left-justified so they can be used in the text (6 X 8 pixels) mode.

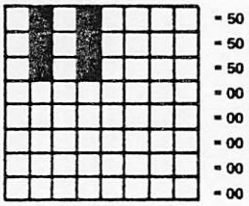
PATTERN 20



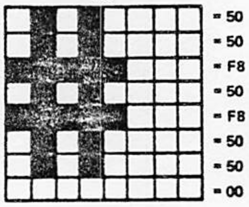
PATTERN 21



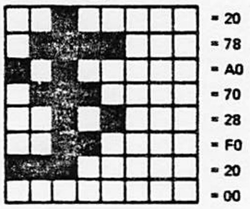
PATTERN 22



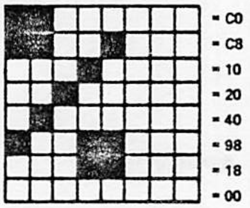
PATTERN 23



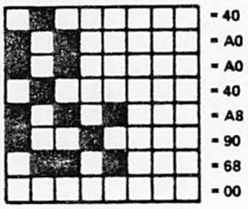
PATTERN 24



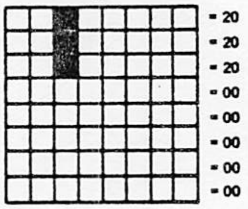
PATTERN 25



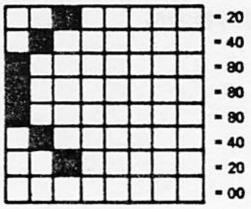
PATTERN 26



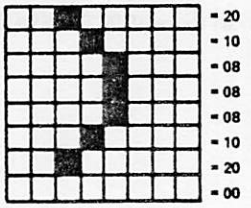
PATTERN 27



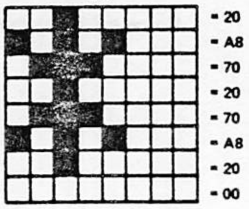
PATTERN 28



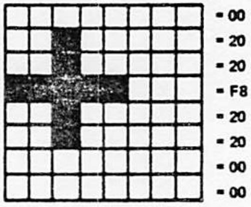
PATTERN 29



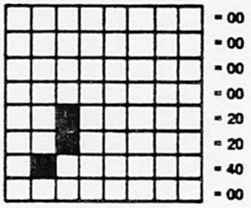
PATTERN 2A



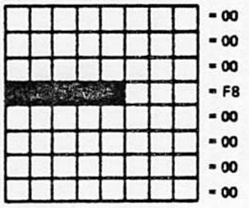
PATTERN 2B



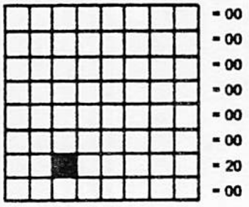
PATTERN 2C



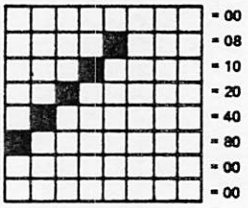
PATTERN 2D



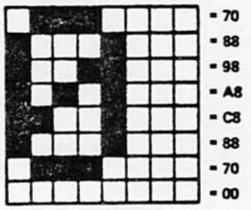
PATTERN 2E



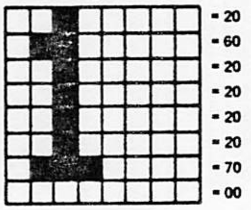
PATTERN 2F



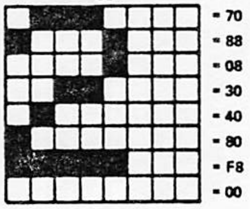
PATTERN 30



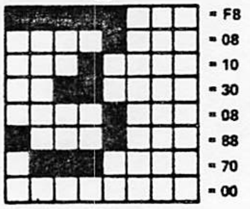
PATTERN 31



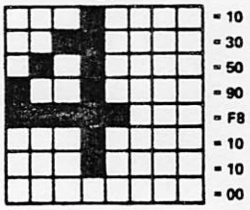
PATTERN 32



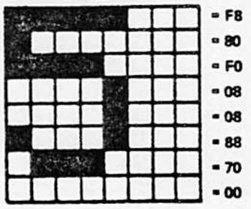
PATTERN 33



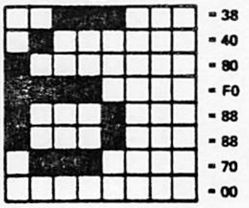
PATTERN 34



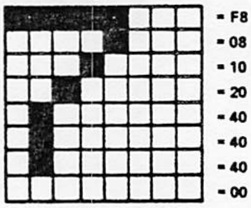
PATTERN 35



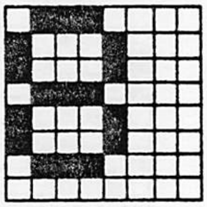
PATTERN 36



PATTERN 37

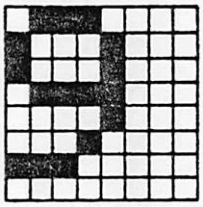


PATTERN 38



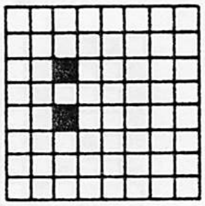
- 70
- 88
- 88
- 70
- 88
- 88
- 70
- 00

PATTERN 39



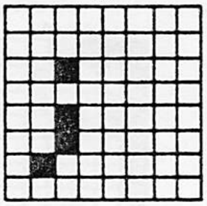
- 70
- 88
- 88
- 78
- 08
- 10
- E0
- 00

PATTERN 3A



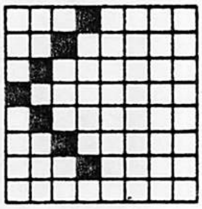
- 00
- 00
- 20
- 00
- 20
- 00
- 00
- 00

PATTERN 3B



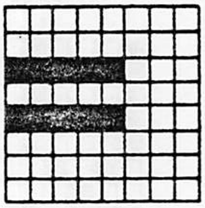
- 00
- 00
- 20
- 00
- 20
- 20
- 40
- 00

PATTERN 3C



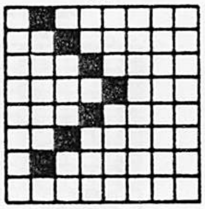
- 10
- 20
- 40
- 80
- 40
- 20
- 10
- 00

PATTERN 3D



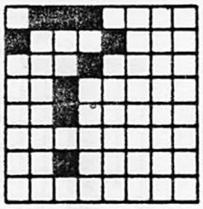
- 00
- 00
- F8
- 00
- F8
- 00
- 00
- 00

PATTERN 3E



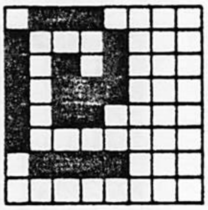
- 40
- 20
- 10
- 08
- 10
- 20
- 40
- 00

PATTERN 3F



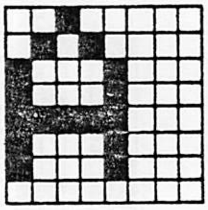
- 70
- 88
- 10
- 20
- 20
- 00
- 20
- 00

PATTERN 40



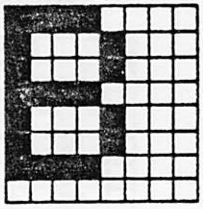
- 70
- 88
- A8
- B8
- B0
- 80
- 78
- 00

PATTERN 41



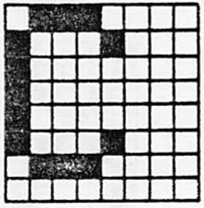
- 20
- 50
- 88
- 88
- F8
- 88
- 88
- 00

PATTERN 42



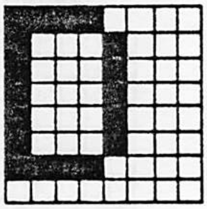
- F0
- 88
- 88
- F0
- 88
- 88
- F0
- 00

PATTERN 43



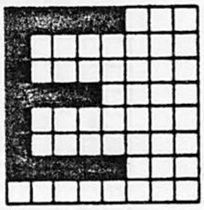
- 70
- 88
- 80
- 80
- 80
- 88
- 70
- 00

PATTERN 44



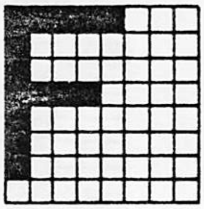
- F0
- 88
- 88
- 88
- 88
- 88
- F0
- 00

PATTERN 45



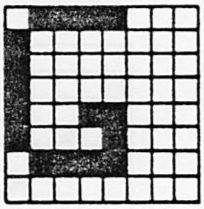
- F8
- 80
- 80
- F0
- 80
- 80
- F8
- 00

PATTERN 46



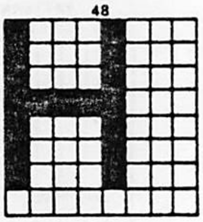
- F8
- 80
- 80
- F0
- 80
- 80
- 80
- 00

PATTERN 47



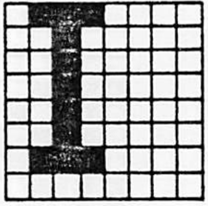
- 78
- 80
- 80
- 80
- 98
- 88
- 78
- 00

PATTERN 48



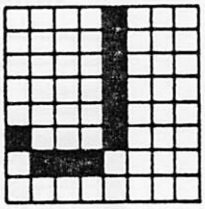
- 88
- 88
- 88
- F8
- 88
- 88
- 88
- 88

PATTERN 49



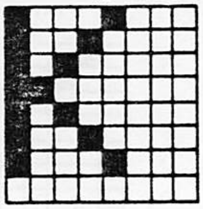
- 70
- 20
- 20
- 20
- 20
- 20
- 70
- 00

PATTERN 4A



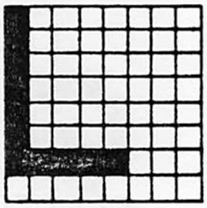
- 08
- 08
- 08
- 08
- 08
- 88
- 70
- 00

PATTERN 4B



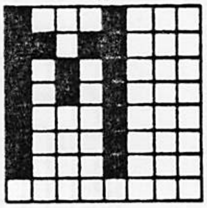
- 88
- 90
- A0
- C0
- A0
- 90
- 88
- 00

PATTERN 4C



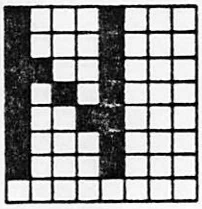
- 80
- 80
- 80
- 80
- 80
- 80
- F8
- 00

PATTERN 4D



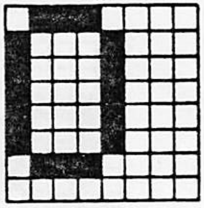
- 88
- D8
- A8
- A8
- 88
- 88
- 88
- 00

PATTERN 4E



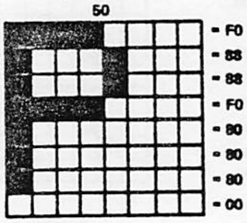
- 88
- 88
- C8
- A8
- 98
- 88
- 88
- 00

PATTERN 4F

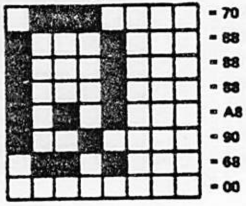


- 70
- 88
- 88
- 88
- 88
- 88
- 70
- 00

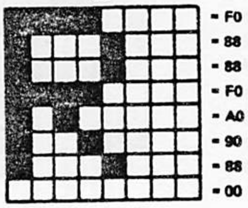
PATTERN
50



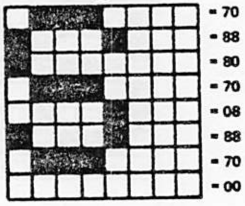
PATTERN
51



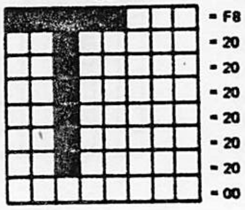
PATTERN
52



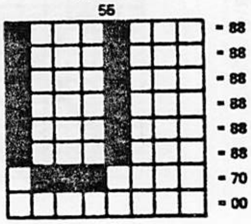
PATTERN
53



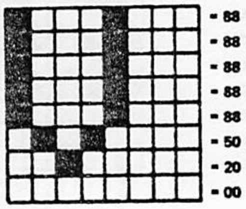
PATTERN
54



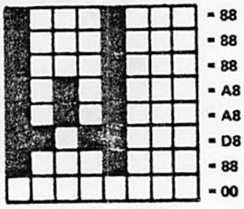
PATTERN
55



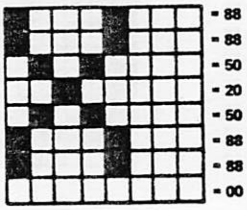
PATTERN
56



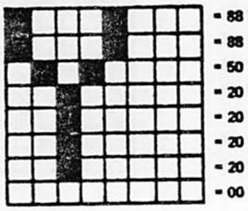
PATTERN
57



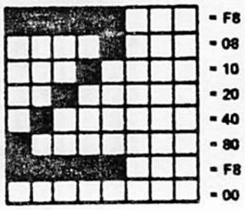
PATTERN
58



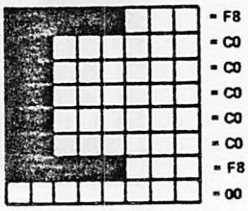
PATTERN
59



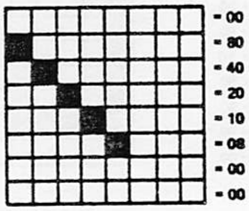
PATTERN
5A



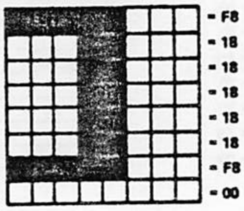
PATTERN
5B



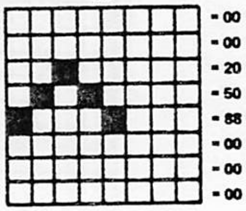
PATTERN
5C



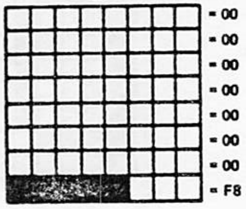
PATTERN
5D



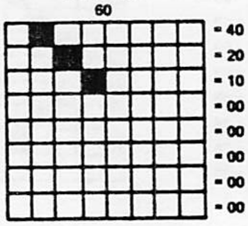
PATTERN
5E



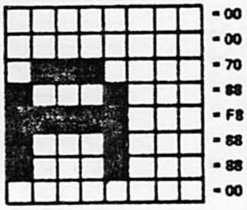
PATTERN
5F



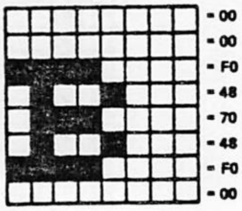
PATTERN
60



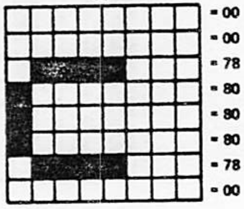
PATTERN
61



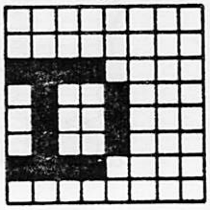
PATTERN
62



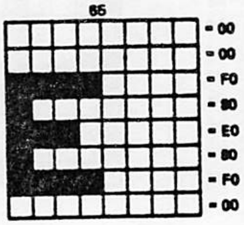
PATTERN
63



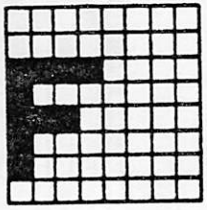
PATTERN
64



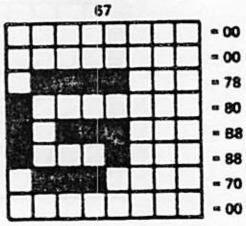
PATTERN
65



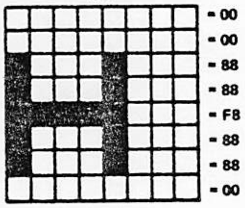
PATTERN
66



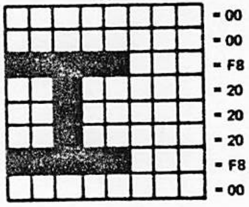
PATTERN
67



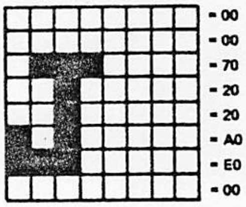
PATTERN 68



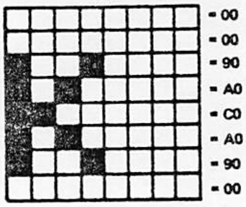
PATTERN 69



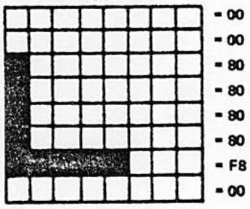
PATTERN 6A



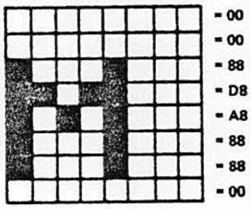
PATTERN 6B



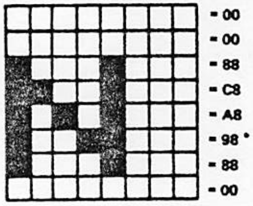
PATTERN 6C



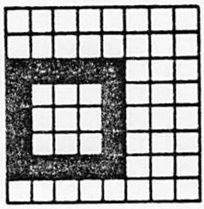
PATTERN 6D



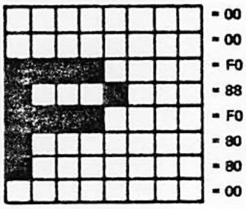
PATTERN 6E



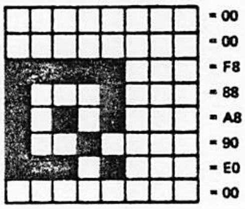
PATTERN 6F



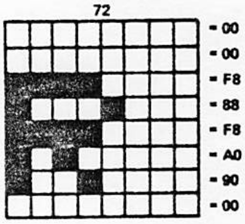
PATTERN 70



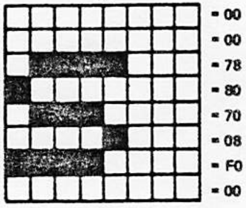
PATTERN 71



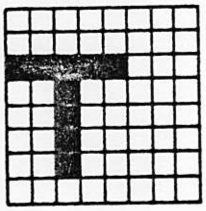
PATTERN 72



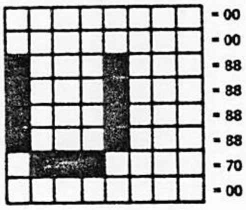
PATTERN 73



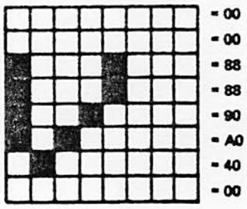
PATTERN 74



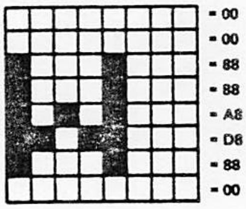
PATTERN 75



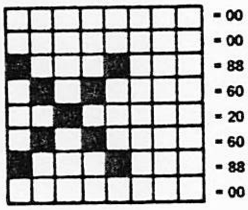
PATTERN 76



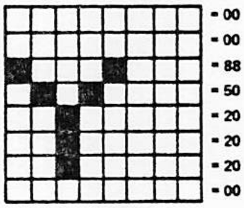
PATTERN 77



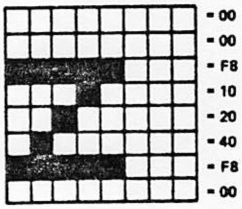
PATTERN 78



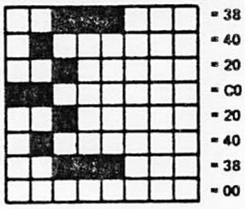
PATTERN 79



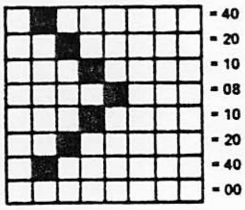
PATTERN 7A



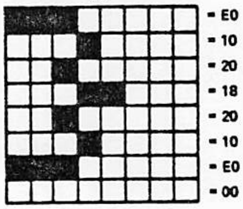
PATTERN 7B



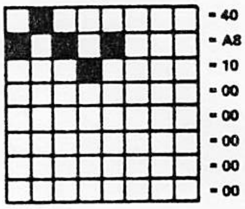
PATTERN 7C



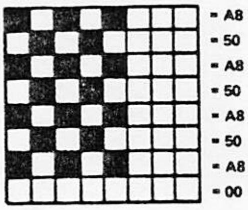
PATTERN 7D



PATTERN 7E



PATTERN 7F



==

AY-3-8910 PROGRAMMABLE SOUND GENERATOR REFERENCE MANUAL

==

PROGRAMMING THE AY-3-8910 PROGRAMMABLE SOUND GENERATOR

GENERAL FEATURES OF THE PSG

Any microprocessor is capable of producing acceptable sounds with only a suitable transducer like a toggle flip-flop connected to a speaker (as is done in the Apple II computer). This method is adequate if the microprocessor has no other tasks to perform while the sound is sustained (since the microprocessor must constantly attend to sustaining the sound output and has little time left over for other tasks). However in arcade style video games, game paddles need to be read, the keyboard needs to be scanned, the video animation needs to be updated, etc. This leaves little time left over for the processor to attend to sustaining or generating sound effects. This is the reason why so many Apple games lack decent sound effects, except for the periods when there is little or no graphics animation taking place. The 6502 microprocessor cannot constantly toggle the speaker port (location $C030 or -16336) to produced a sustained sound output while simultaneously updating the graphics data in RAM to create animation. You can either have animation or sound effects but not both at the same time.

The General Instruments AY-3-8910 Programmable Sound Generator (PSG) integrated circuit not only solves this problem, but considerably extends the range of sound effects which can be generated. The PSG is software programmable so that the host processor (here the 6502) need only pass a few command bytes to the PSG and then move on to perform other tasks. The PSG will generate sounds continuously until told to stop by the host processor. In fact the PSG acts somewhat like a co-processor working independently of the 6502 microprocessor so that the work load on the 6502 is lightened considerably for arcade game or music generation programs.

The PSG has three independent software-controlled tone generators. Each has a 9-octave range from a low frequency of 31.23 Hz to an ultrasonic high frequency of 127,875 Hz. In addition each tone generator has an independent 16-level volume control with a range from 0 (no sound) to 15 (maximum volume). The output of these tone generators is a pulse waveform with a 50% duty cycle - i.e. a square wave. The output of these three tone generators may be combined to produce complex multi-tonal sound effects or multi- voice musical compositions.

The PSG has one pseudo-random noise generator with a frequency range from 4,000 Hz to 125,000 Hz. The noise generator is useful for creating such sound effects as gunshots, steam whistles, explosions. hisses, wind sounds, breaking waves, hand claps, drums, cymbals, tambourines, etc.

There are three channels or sound outputs on the PSG. These channels are designated channel A, channel B and channel C. Each channel consists of two parts: the tone generator and the noise generator. Three independent tone generators make up the tone generator portions of the channels; while the single noise generator makes up the noise signal portion of the channels. Each channel has an independent volume control which manipulates the level of both the tone generator signal and the noise generator signal. These 16-level volume controls are the same as those mentioned above in the description of the tone generators. Strictly speaking, neither the tone generators nor the noise generators have their own volume controls. The channels (tone and noise taken together) have volume controls. Each channel may be all tone and no noise, all noise and no tone. or both tone and noise. Although you can have three different frequencies of tones on the three channels, the noise generator frequency must be the same for all three channels since there is only one noise generator.

One of the more powerful features of the PSG is the automatic envelope control which allows you to control the volume of a tone over a period of time. For example, a violin string when bowed has a rather constant volume; but when plucked, the volume starts off loud and then gradually dies down. The graph of volume versus time for a sound is called the envelope of the sound. Most musical instrument envelopes have a rise time (usually very fast) where the volume starts from zero and reaches some maximum level. This is followed by fast decay to a relatively constant lower level of volume, which gradually diminishes until the volume reaches zero. Instruments such as the piano, harpsichord, banjo, chimes, xylophone, guitar, etc. have this type of envelope. On the other hand instruments such as the organ have an envelope which is relatively constant while the organ key is held down and then dies rapidly when the key is released. The automatic envelope control feature can be used to simulate the envelopes of various musical instruments. It is also useful in sound effects such as gunshot sounds where instead of a tone you have noise with a rapidly decaying volume level (or envelope).

PSG I/O PORT ADDRESSES:

The PSG is controlled by writing data to two I/O ports on the Arcade Board whose addresses depend upon then slot in which the Arcade Board is located. These two ports are known as the Register Select Port and the Data R/W (for Read/Write) Port. Their addresses are given in the table on the next page for the various slots.

PROGRAMMING THE AY-3-8910 PROGRAMMABLE SOUND GENERATOR

SLOT #	REGISTER SELECT PORT		DATA R/W PORT	
1	$C095	-16235	$C096	-16234
2	$C0A5	-16219	$C0A6	-16218
3	$C0B5	-16203	$C0B6	-16202
4	$C0C5	-16187	$C0C6	-16186
5	$C0D5	-16171	$C0D6	-16170
6	$C0E5	-16155	$C0E6	-16154
7	$C0F5	-16139	$C0F6	-16138

PSG I/O PORT ADDRESSES FOR SLOTS 1 THRU 7 IN HEX AND DECIMAL

A general formula can be used to calculate the port addresses in hexadecimal. It is:

```
Register Select Port  =  $C0N5
Data R/W Port         =  $C0N6
```

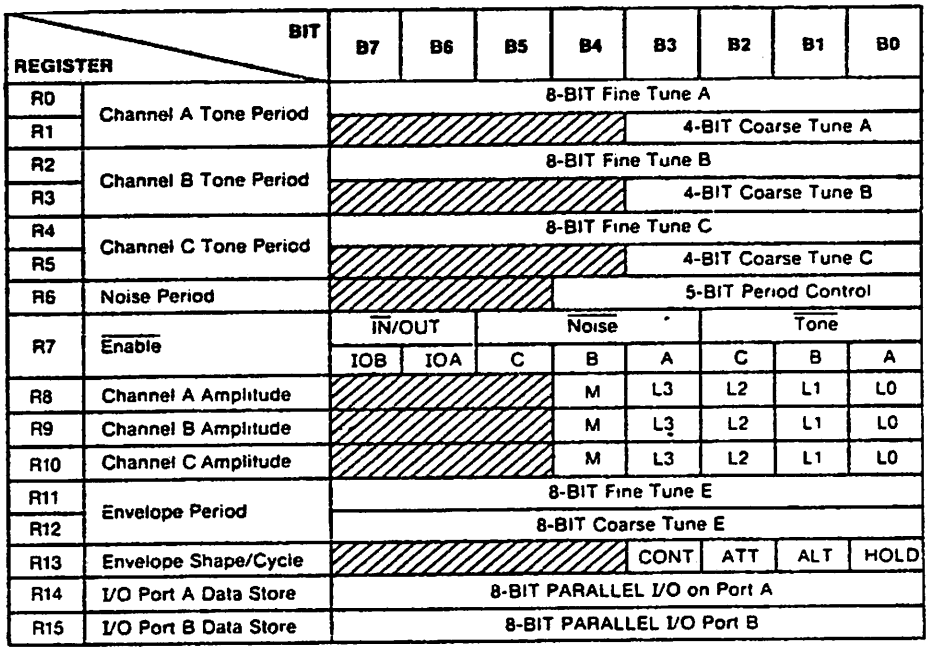
Where N = the slot number (1 - 7) plus 8 in hexadecimal.

PSG CONTROL REGISTERS

The PSG has 16 internal registers which control the operation of the PSG. To program the PSG to create a certain sound, you must send data to these various registers. The general sequence consists of first sending the register number (0 thru 15) to the Register Select Port and then sending the desired data byte (0 thru 255) to the Data R/W Port. Once a certain register has been selected (by sending the register number to the Register Select Port), you may send multiple bytes of data to that register via the Data R/W Port without having to specify the register number each time. For example, you may wish to send several values to register 1 in sequence. You may send a 1 to the Register Select Port, then the first byte of data to the Data R/W Port, then a 1 again to the Register Select Port, followed by the second byte of data to the Data R/W Port, followed by a 1 again to the Register Select Port, etc. Or you may simply set up the register address by sending a 1 to the Register Select Port only once. From then on just send the date bytes in an uninterrupted sequence to the Data R/W Port. There is no need to repeatedly send a 1 to the Register Select Port before sending the next byte of data to the Data R/W Port. Of course if you are going to send data to a new register number, you must set up the PSG to receive the data in the correct register by first sending the desired register number to the Register Select Port. Once the register is set up, you can continue sending data to the Data R/W Port for as long as you wish.

The Register Select Port is a write-only port. If you forget what register number the PSG is set up for, you can not read the Register Select Port to find out. You will have to set up the PSG for whatever register number to which you wish to send data.

The Data R/W Port is a read/write port. You may send data to that port as well as read it back again; i.e. You can read the contents of all the registers as well as write to them. Typically you would use the BASIC POKE instruction to send data to any given port and the BASIC PEEK instruction to read data from any given port. In 6502 assembly language, the corresponding instructions would be "STA port address" and "LDA port address". The functions of the various registers are given below:



REGISTER	BIT	B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A Tone Period	8-BIT Fine Tune A							
R1		▨	▨	▨	▨	4-BIT Coarse Tune A			
R2	Channel B Tone Period	8-BIT Fine Tune B							
R3		▨	▨	▨	▨	4-BIT Coarse Tune B			
R4	Channel C Tone Period	8-BIT Fine Tune C							
R5		▨	▨	▨	▨	4-BIT Coarse Tune C			
R6	Noise Period	▨	▨	▨	5-BIT Period Control				
R7	$\overline{\text{Enable}}$	$\overline{\text{IN}}$/OUT		$\overline{\text{Noise}}$			$\overline{\text{Tone}}$		
		IOB	IOA	C	B	A	C	B	A
R8	Channel A Amplitude	▨	▨	▨	M	L3	L2	L1	L0
R9	Channel B Amplitude	▨	▨	▨	M	L3	L2	L1	L0
R10	Channel C Amplitude	▨	▨	▨	M	L3	L2	L1	L0
R11	Envelope Period	8-BIT Fine Tune E							
R12		8-BIT Coarse Tune E							
R13	Envelope Shape/Cycle	▨	▨	▨	▨	CONT	ATT	ALT	HOLD
R14	I/O Port A Data Store	8-BIT PARALLEL I/O on Port A							
R15	I/O Port B Data Store	8-BIT PARALLEL I/O Port B							

PSG CONTROL REGISTER FUNCTIONS

There are two general purpose parallel I/O ports implemented on the PSG. These I/O ports are addressed as PSG registers 14 and 15. Each can be either input or output ports independent from each other, however, neither can be both input and output at the same time. Whether they are input or output ports depends upon the two most significant bits of the mixer control register, register 7. Register 14 (I/O Port A) is used to control the 16 programmable filters provided on the Arcade Board. These may be used to further extend the range of sounds capable of being generated by the PSG. These programmable filters are NOT a part of the PSG, but were added as an additional feature by the designers of the Arcade Board. See below for more details on the use of the programmable filters. I/O Port B or register 15 is unused on the Arcade Board.

TONE FREQUENCY CONTROL REGISTERS (0 - 5)

The frequencies which can be produced by each of the three tone generators are controlled by the tone frequency control registers 0 thru 5. Registers 0 and 1 control the tone generator frequency for channel A, registers 2 and 3 control the tone generator frequency for channel B, and registers 4 and 5 control the tone generator frequency for channel C. Each register pair is concatenated to form a single 12-bit register for each of the three channels. The lower numbered register of the pair (registers 0, 2, and 4) make up the lower order 8 bits of the 12 bit number. This register is called the Fine Tune Control register. The higher numbered register of each pair (registers 1, 3, and 5), is only 4 bits long, and makes up the high order 4 bits of the 12 bit value used to specify the frequency of the given tone. This register is therefore called the Coarse Tune Control register. Since 12-bits allows for the specification of 4096 different binary numbers, there are a maximum of 4096 different frequencies ossible for all three channels. Therefore only certain discrete frequencies can be selected; you can not have all possible frequencies withing the 9-octave range of the tone generators, as this would require an infinite number of bits in the frequency control registers.

The PSG produces tone frequencies by taking the input clock frequency and first dividing it by 16. This base frequency is then further divided by the 12-bit value formed by concatenation of the two tone frequency control registers for each channel. The input clock frequency is 2.046 MHz. Di vided by 16 gives a base frequency of 127,875 Hz. Since division by 0 is not allowed, the lowest value allowed for the 12-bit divider value is 1, even if both of the two registers in the tone frequency control registers for a given channel are 0. This means that the highest allowed frequency is the base frequency divided by 1 or 127,875 Hz and the lowest allowed frequency is the base frequency divided by 4095 (4095 is the highest number that can be specified with only 12 bits) or 31.23 Hz. Therefore the lower the value specified by the tone frequency control register pair for a given channel, the higher is the frequency and vice-versa. The 12-bit number represented by the tone frequency control register pair is actually proportional to the tone period rather than to the frequency.

To calculate which values to enter into each register of the pair of tone frequency control registers for a given channel, and a certain desired frequecncy, use the following method.

127,875 Hz / Desired frequency = 12-bit value + remainder

Since more often that not the answer will not be an integer, you will have to chose between rounding up or rounding down to the next integer value. You can not send fractional values to the tone frequency control registers. Whether you round up or down depends on which integer value gives a final frequency closest to the desired frequency. For example suppose you want a desired frequency of 440 Hz, which is the note A just above middle C in the equal-tempered scale. You would divide 127,875 by 440 to get:

127,875 / 440 = 290.63

Well, we must choose either 290 or 291 since we must send an integer value to the tone frequency control registers. If we chose 290 we would get a actual frequency of:

127,875 / 290 = 440.95

If we chose 291 we would get an actual frequency of:

127,875 / 291 = 439.43

In this case chosing 291 would give us the closest frequency to the desired frequency of 440 Hz. In this example, however, it would not make much difference what value we chose since both values would give actual frequencies so close to the desired 440 Hz that most people would not be able to tell the difference.

We still are not finished, though. This number from the above formula gives us the 12-bit value necessary to give the closest approximation to the desired frequency. Since the 6502 is an 8-bit microprocessor and because the two registers that make up the tone frequency control registers are 8 bits (fine tune control register) and 4 bits (coarse tune control register) we must break up our 12 bit answer into a the 4 most significant bits (the 4 highest order bits) and the 8 least significant bits (the 8 lowest order bits). The 4-bit value will be sent to the Coarse Tune Control register and the 8-bit value will be sent to the Fine Tune Control register. This is easily done by the following method:

To calculate the Coarse Tune register value (4 MSB's):

Calculate: 12-bit value / 256 and take the integer part only.

To calculate the Fine Tune register value (8 LSB's):

Calculate: 12-bit value - 256 * (4 MSB's)

For example, above we arrived at the best 12-bit value of 291. Divide 291 by 256 and take the integer portion gives us 1. We ignore the remainder. Therefore the coarse tune register value is 1. To find the fine tune register value we multiply 1 by 256 and subtract the answer from 291 to get 35. This is the fine tune register value.

NOISE FREQUENCY CONTROL REGISTER

Register 6 controls the "frequency" of the noise generator. This register is only 5 bits wide so therefore only values from 0 to 31 can be sent to it. However a value of 0 is treated as equal 1 since division by 0 is not allowed, and the noise frequency is obtained by dividing the 2.046 MHz input clock by 16 and then dividing by the value in the noise frequency control register. Therefore the highest noise frequency possible is 2,046,000 / 16 = 127,875 Hz and the lowest frequency is 2,046,000/ 16 further divided by 31 = 4125 Hz.

MIXER CONTROL REGISTER

Register 7 controls the mixer which determines what channels will go to the sound output of the PSG. Each of the 8 bits of this register has its own special function, whether to enable or disable tone or noise for each channel, or to determine the direction of the two parallel I/O ports of registers 14 and 15. The function of the 8 bits are given below:

Bit #	Function if = 0	Function if = 1
0	Enable tone on channel A	Disable tone on channel A
1	Enable tone on channel B	Disable tone on channel B
2	Enable tone on channel C	Disable tone on channel C
3	Enable noise on channel A	Disable noise on channel A
4	Enable noise on channel B	Disable noise on channel B
5	Enable noise on channel C	Disable noise on channel C
6	I/O port A is input port	I/O port A is output port
7	I/O port B is input port	I/O port B is output port

FUNCTION OF CONTROL BITS OF MIXER CONTROL REGISTER

For those who have problems with binary numbers a few common configurations are given below:

255 = Disable tone and noise on all channels, both I/O ports are output ports. PSG essentially turned off.
254 = Enable tone on channel A, but no noise, both I/O ports are output ports.
253 = Enable tone on channel B, but no noise, both I/O ports are output ports.
252 = Enable tone on channels A and B, but no noise, both I/O ports are output ports.
251 = Enable tone on channel C, but no noise, both I/O ports are output ports.
248 = Enable tone on all channels, but no noise, both I/O ports are output ports.
247 = Enable noise only on channel A, both I/O ports are output ports.
240 = Enable tone on all channels and noise on channel A only, both I/O ports are output ports.
239 = Enable noise only on channel B, both I/O ports are output ports.
224 = Enable tone on all channels and noise on channels A and B, both I/O ports are output ports.
223 = Enable noise only on channel C, both I/O ports are output ports.
199 = Enable noise on all channel, but no tone, both I/O ports are output ports.
192 = Enable tone and noise on all channels, both I/O ports are output ports.
128 = Enable tone and noise on all channels, I/O port A is input port and I/O port B is output port.
0 = Enable tone and noise on all channels, both I/O ports are input ports.

For example, if you wished to turn off the PSG so that there would be no sound output whatsoever, you would send 255 to register 7. If you only wanted to turn on tone for channel A but keep everything else off, you would send 254 to register 7. If you wanted all channels to have both tone and noise, then you would send 192 to register 7. Normally you would keep both I/O ports as output ports and these would never be changed.

VOLUME CONTROL REGISTERS

Registers 8, 9, and 10 control the volumes for channels A, B, and C respecitively. These registers control both the tone and noise signals (if enabled by register 7) for their respective channels. These registers are 5 bits wide, but bit 4 is special. Normally you would send a value from 0 to 15 to any of these registers. A value of 15 corresponds to the maximum volume for the channel. whereas a value of 0 corresponds to 0 volume for the channel. It is best, however, to turn off a channel using register 7, rather than by turning the volume to 0. Sending a value of 16 to 31 to either of the volume control register sets that channel to the automatic envelope control mode. That is whenever bit 4 is set then the automatic envelope control determines the volume of the sound coming out of the channel controlled by the particular volume control register in question. This allows for a whole new range of special effects.

A very interesting trick is to vary the volume of a channel's sound output in a predetermined manner by sending different values from 0 to 15 to the appropriate volume control register. In this way you could "manually" create any envelope shape that you desired. The advantage of this method of envelope control over the automatic envelope control is that you can create practically any envelope shape you desire, whereas the automatic envelope control gives you only a limited number of envelope shapes to chose from. However, the automatic envelope control does not require constant attention by the 6502, whereas the software driven envelope control method does.

AUTOMATIC ENVELOPE CONTROL REGISTERS

Registers 11, 12, and 13 are used to control the parameters for the automatic envelope control capability of the PSG. There are two factors to consider in selecting a particular envelope for the sound output. First, there is the "envelope period" which is the length of time over which the envelope extends. Second there is the overall shape of the envelope. For example, suppose we have an envelope that starts off at the maximum volume and decays linearly to zero volume. The envelope shape is shown below. This same envelope shape can be stretched out over a long period of time (second figure) or it can constrict over a much shorter period of time (third figure). The envelope period is the duration of the envelope shape. For this particular envelope shape a period of a few seconds would produce a "bell like" decay where the sound level gradually dies down.

An envelope period of only a few tenths of a second would produce a metallic "chink-like" sound for the exact same envelope shape.

```
TIME ---->

*                   *                                   *
* *                 *    *                              **
*   *               *         *                         * *
*     *             *              *                    *  *
*       *           *                   *               *   *
*         *         *                        *          *    *
*           *       *                             *     *     *

BASIC SHAPE             LONG ENVELOPE PERIOD       SHORT PERIOD
```

Registers 11 and 12 specify the envelope period. These two registers are each 8 bits wide. They are concatenated into a single 16 bit value with the low 8 bits coming from register 11 and the high 8 bits coming from register 12. Register 11 is called the Envelope Fine Tune Register and register 12 is called the Envelope Coarse Tune Register. The length of time for the envelope period is determined by dividing the input clock frequency of 2.046 MHz first by 256 and then again by the 16 bit value from concatenating registers 11 and 12. This value is a frequency value; so the period is just the inverse of the frequency, that is 1 / frequency. As in previous cases, division by 0 is illegal so a value of 0 in both registers 11 and 12 is treated as being equal to 1. The longest period is obtained when the 16 bit envelope period value has its maximum value of 65535 and the shortest period is when the envelope period is equal to 1.

For example, suppose we wish to have an envelope period of about 2 seconds to simulate a chime-like envelope shape. We would use the following formula to calculate the 16 bit envelope period value which would give us an envelope period of 2 seconds.

16 bit Value = (Input Clock Frequency / 256) * Desired Period

16 bit Value = (2,046,000 / 256) * 2 = 15984.38

Therefore we would need to have a 16 bit envelope period value of 15984 (ignoring the fraction) to get an envelope period of about 2 seconds. We still have to break this 16 bit value into two 8 bit values for registers 11 and 12. This is easily done as follows:

Register 12 Value = INT (16 bit Value / 256)
(where INT means to take the interger portion only)

Register 11 Value = 16 bit Value - (256 * Register 12 Value)

In our example the 16 bit value is 15984. The register 12 value is then:

Register 12 Value = INT (15984 / 256) = 62

The register 11 value is then:
Register 11 Value = 15984 - 256 * 62 = 112

If we send a value of 112 to register 11 and a value of 62 to register 12, we should get an envelope period of about 2 seconds.

We can obtain a range of envelope periods from approximately 8 seconds maximum to a minimum of 0.00013 seconds.

Register 13 controls the overall envelope shape. This register is only 4 bits wide. It can have any integer value from 0 to 15. Each of the 4 bits controls the function listed below:

Bit 1	-	Hold
Bit 2	-	Alternate
Bit 3	-	Attack
Bit 4	-	Continue

HOLD - when set to 1 limits the envelope to one cycle. That is, it does not repeat. At the end of the envelope period, the volume level is held constant at either the maximum volume (15) or the minimum volume (0), depending upon whether the Attack bit was set to 1 or 0 respectively.

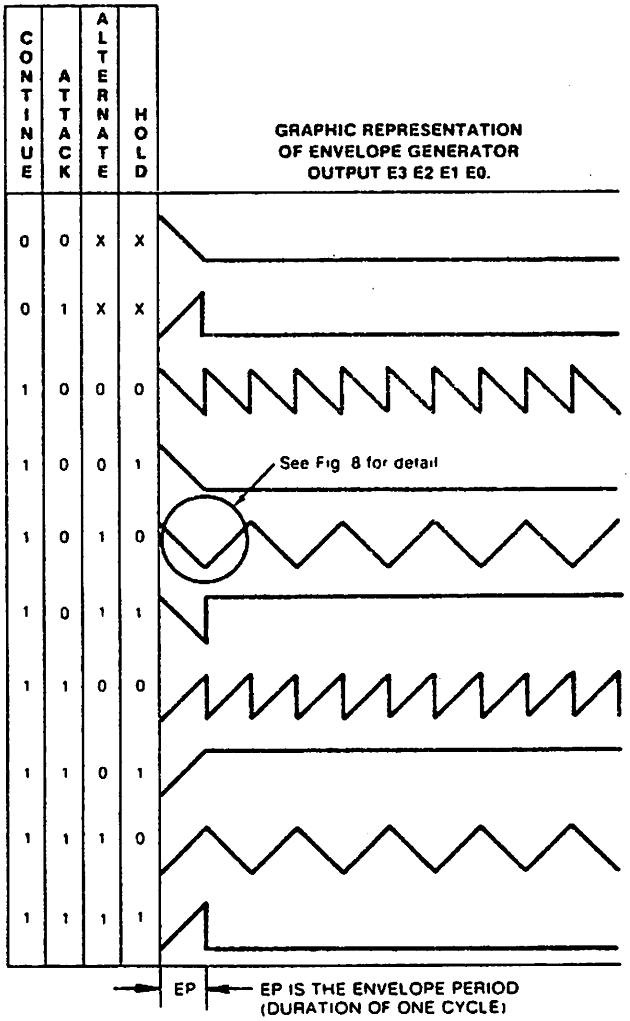
ALTERNATE - when set to 1 the volume alternates between the maximum and the minimum volume. This occurs at the end of each envelope period.

ATTACK - when set to 1 the envelope shape begins as a 0 volume and increases to the maximum volume of 15 - this is the attack portion of the envelope. When set to 0 the envelope shape starts at the maximum volume of 15 and decays to 0 volume - this is the decay portion of the envelope.

CONTINUE - when set to 1 the envelope pattern will be as defined by the Hold bit. When set to 0 the the envelope's volume will be set to 0 at the end of the envelope period and will be held at 0. This bit is used to repeat the envelope shape over and over as long as the Hold bit is set to 0.

The chart below illustrates the effects of the various control bits:

Note the triagular envelopes actually represent two envelope periods. The first period is in the rise time to maximum volume (since the alternate bit is set, the volume decays to 0 during the second envelope period). The envelope has an effective period of twice the time designated by the contents of registers 11 and 12.



CONTINUE	ATTACK	ALTERNATE	HOLD
0	0	X	X
0	1	X	X
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

GRAPHIC REPRESENTATION OF ENVELOPE GENERATOR OUTPUT E3 E2 E1 E0.
See Fig 8 for detail
EP
EP IS THE ENVELOPE PERIOD (DURATION OF ONE CYCLE)

PROGRAMMABLE FILTERS

The PSG has considerable sound generation capability. However, its output is limited to pulse waveforms. To further extend the range of sounds capable of being generated, the designers of the Arcade Board have provided a set of 16 software-selectable filters which are used to modify the output sounds of the PSG. These filters are selected by the low 4 bits of I/O port A of the PSG, that is, register 14. This I/O port must be set as an output port by setting bit 6 of the mixer control register (register 7) to 1. There are basically 4 types of filters, each with 4 different cutoff or center frequencies. These filter types are: high-pass, low-pass, band-pass, and notch- pass. The sound outputs of the PSG are not simple sine waves. They consist of a whole range of frequencies which are integral multiples of the fundamental frequency. This frequency is determined by the settings of the tone frequency control registers. To understand this further, you would need to know about the physics of sounds, harmonics, overtones, etc. which are not the subject of this manual. To simplify things, we will just say that the filters modify the sounds much like the bass and treble tone controls on a stereo amplifier.

Basically, the high-pass filter modifies the PSG sound output by emphasizing the higher frequency sounds and harmonics and by attenuating (lowering the relative volume) the lower frequency sounds and harmonics. The effect of passing the PSG sound output through a high pass filter is similar to turning the tone control of an amplifier to more treble. There are 4 different high-pass filters to choose from. Each has a different cutoff frequency. The higher the cutoff frequency, the more "brilliant" is the sound as less and less of the lower harmonics are allowed to pass through. The cutoff frequencies are about 400 Hz, 800 Hz, 1600 Hz, and 3200 Hz. The high-pass filters are used to simulate brilliant sounding musical instruments such as strings, and violins.

The low-pass filter is just the opposite of the high-pass filter as it emphasizes the lower frequency harmonics by attenuating the higher frequency harmonics. Its effect is like turning the tone control of an amplifier to more bass. As with the high pass filters, there are 4 cutoff frequencies to choose from; 400 Hz, 800 Hz, 1600 Hz, and 3200 Hz. The low-pass filters are used to simulate more mellow sounding musical instruments such as the flute and flute-like organ voices.

The band-pass filters selectively emphasize frequencies around some central frequency while attenuating those on either end of it. This selects a certain band of frequencies while attenuating all others. There are 4 different band-pass filters to choose from - each with a different center frequency (Note: for high and low-pass filters, we speak of the cutoff frequency; while for band and notch-pass filters, we speak of the center frequency). The center frequencies are at 400 Hz, 800 Hz, 1600

Hz, and 3200 Hz. The band-pass filters are used to simulate horn-type sounds.

The notch-pass filters are just the opposite of the band-pass filters. They reject a band of frequencis around some center frequency while passing all others. Like the band-pass filters, there are 4 cutoff frequencies at 400 Hz, 800 Hz, 1600 Hz. and 3200 Hz. The notch-pass filters are not used to simulate any particular musical instrument although at times, they exhibit a reedy sound similar to bagpipes.

To select a particular filter and center/cutoff frequency you must first have established register 14 to be an output port by setting bit 6 in register 7 to a 1. This is usually done when enabling tones and noise on the various channels A, B, and C. Once register 14 is setup as an output port, the low order 4 bits are used to select the desired filter. The function of the bits is:

Bits 0 and 1	-	Select the filter type		
		00	-	High-pass
		01	-	Low-pass
		10	-	Band-pass
		11	-	Notch-pass
Bits 2 and 3	-	Select the cutoff/center frequency		
		00	-	400 Hz
		01	-	800 Hz
		10	-	1600 Hz
		11	-	3200 Hz

For those who do not understand binary:

Value in Register 14	FilterType	Cutoff/CenterFrequency
0	High-pass	400 Hz
1	Low-pass	400 Hz
2	Band-pass	400 Hz
3	Notch-pass	400 Hz
4	High-pass	800 Hz
5	Low-pass	800 Hz
6	Band-pass	800 Hz
7	Notch-pass	800 Hz
8	High-pass	1600 Hz
9	Low-pass	1600 Hz
10	Band-pass	1600 Hz
11	Notch-pass	1600 Hz
12	High-pass	3200 Hz
13	Low-pass	3200 Hz
14	Band-pass	3200 Hz
15	Notch-pass	3200 Hz

NOTE: Due to the unusual manner in which Applesoft and Integer BASIC execute the POKE instruction (by first reading the location to be POKEd and then writing the data to it), there is a small problem with using the BASIC POKE instruction when selecting filters. This is because register 14 becomes strictly write-only once it is designated as an output port. Occasionally when you use the BASIC POKE instruction to select a filter, the wrong data will be sent to register 14 and the wrong filter will be selected. To ensure that this never happens, you may want to use a machine language store accumulator instruction to send data to register 14. This problem does not occur with all the other registers on the PSG, so it is safe to use the BASIC POKE instruction when writing to them.

SPECIAL TRICKS USING THE PSG

ADSR Envelopes:

The automatic envelope control feature is very powerful and easy to use, though a bit limited. True you can construct composite envelopes where you use an envelope shape that starts from 0 volume and rises and holds at the maximum volume of 15, then after waiting a suitable period you could then use a decaying envelope shape that starts at maximum volume and decays to 0. In this manner you could get more complex envelopes than the ones mentioned in the table above in the automatic envelope control section. However this method still does not allow for envelopes whose shape may rise to maximum volume, then decay to some intermediate volume, hold there for a while, then decay to 0. This type of envlope is more like that of most musical instruments. This is known as ADSR envelope control, where you specify the Attack (how fast maximum volume is reached), the Decay (how fast it decays to a specified intermediate volume level), the Sustain (how long the volume is sustained at the intermediate volume level), and the Decay (how long it takes to decay to 0 volume). This kind of evelope could not easily be done using the automatic envelope control. You could, however, create this kind of envelope "manually" by use of the software-driven envelope control method, where your software specifes the changes in volume level over time to create the desired envelope shape. You could create virtually any envelope shape with this method.

Tremolo:

The above method works well to create any ADSR type envelope or any weird envelope you want. If you were to oscillate the volume over a small range with time you could obtain a tremolo effect. For instance if the note you chose had a volume of 8, you could alternate its volume from 7 to 8 to 9 to 8 to 7 etc. This is amplitude modulation.

Vibrato:

Similar in concept and sound to tremolo is vibrato, where instead of the volume oscillating with time the frequency of the note oscillates instead. You could achieve this effect by changing the value in the tone frequency control register(s) in an oscillatory manner.

Beats and Chimes:
When two notes are played simultaneously with slightly different frequencies, you will hear beat tones. This effect can be used to create chime or bell type sounds. To create a chime sound you would have two tones which differ in frequency by only a small amount. One of the tones would use the automatic envelope control feature using an envelope which starts off at the maximum volume and decays slowy to 0 volume. The envelope period should be around 4 seconds. The other tone should be at a constant volume but not too loud, say about 4 or 6. As one tone starts to decay in loudness the other constant-volume tone becomes more noticeabel. The beats created simulate the ringing effect of chimes.

Octave Shifts:
In music the relationship between a given note and the same note an octave lower or higher is that the frequencies are related exactly two to one. That is to get the frequency of a note an octave higer, you would just multiply the frequency by two. For example the note A above middle C is A 440 Hz. The note A one octave lower is A 220 Hz and the note A an octave lower is A 880 Hz. You can easily make a given note an octave lower or higher by merely multiplying or dividing the 12 bit tone period value given by the coarse and fine tune registers of the tone control registers. For example the 12 bit value for A 440 Hz was earlier found to be 291. To get A 880 Hz we would divide this value by 2 to get 145.5. We must chose either 145 or 146, since the value must be an integer. Remember the values in the tone frequency control registers are period values and not the frequency - the period is the inverse of the freequency. Therefore the lower the period the higher is the frequency and vice-versa. To get A 220 Hz we just multiply 291 by 2 to get 582. Quite pleasing effects can be achieved by playing a note together with the same note an octave higher. This adds brilliance to the sound, much like an organ sound. You could also try playing the note along with the same note an octave higher and also the same note two octaves higher (divide the period value by 4).

4 Bit DAC Usage:
The volume controls on the PSG are actually 4 bit digital to analog converters. They can be used to approximate almost any sound by a process known as amplitude modulation, the same process used in AM radio broadcasting. The basic technique is exactly the same as the software-driven envelope control technique mentioned above with two differences. First of all the software-driven envelope control method varied the volume of an audible sound, that is a sound in the frequency range of 35 Hz to about 20,000 Hz. Second the period of this envelope is quite long, on the order of seconds. In the Amplitude Modulation technique the sound whose volume is being changed or modulated by software is ultrasonic - that is you can not hear it. This is known as the carrier wave in AM radio broadcasting talk. Also the period of the software-created "envelope" is in the audio frequency range. With this technique you can construct almost any waveform, even human speech. This requires constant

processor attention, even more than the software-driven envelope control method, as the periods involved are much shorter. In order to understand this method better an example is given. If you set the mixer control register to enable tone only on channel A and the tone frequency control registers for channel A (registers 0 and 1) are set to 1 and 0 respectively, you would hear no sound as the frequency produced is far beyond the audio range. No matter what volume (0 to 15) you set the volume control register for channel A (register 8) to you would hear nothing. But now suppose you set the volume to 0 then waited for 2.27 milliseconds and then set the volume to 15 and waited for 2.27 milliseconds again and then set the volume to 0 again and repeated the process. What you would hear is a square wave of frequency 440 Hz. You could instead gradually change the volume from 0 to 15 in the same 2.27 milliseconds and ramp it down from 15 to 0 again in the next 2.27 milliseconds. Then you would hear a staircase or ramp waveform of frequency 440 Hz. If you passed this through one of the low-pass filters, the sharp edges would be smoothed out and it would sound more like a triangle wave. You could vary the volume in any way you chose to get all kinds of different sounds. To synthesize human speech you would have to know just how the volume of the speech varied with time. This could be rather difficult to find out without the use of some fancy equipment and some analog to digital converters.

USING THE 60 HZ INTERRUPT AS A TIMER FOR MUSIC

The VDP chip generates an interrupt every 60th of a second. This could be used to keep track of the elapsed time for timing such things as the duration of notes in a musical composition or the tempo for the composition. 64 is an easy number to work with as it is divisible by 2 for half notes, by 4 for quarter notes, by 8 for eighth notes, by 16 for sixteenth notes, and by 32 for thirty-second notes. You would use the interrupt to increment some memory value from 0 to 63 to use as a timer. Of course you would have to write this interrupt routine in 6502 assembly language. The use of interrupts is not for the BASIC programmer. Apart from the use of the interrupt to time the duration of notes it can be used to time and coordinate the occurrence of events. For instance you may want to use the timer as a timed fuse for a bomb in a game, where after a certain condition is met the bomb's fuse is activated. The player has only a certain amount of time to do something before the bomb explodes. There are all kinds of uses for this interrupt timer.

OPERATION

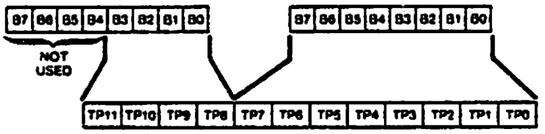
Since all functions of the PSG are controlled by the host processor via a series of register loads, a detailed description of the PSG operation can best be accomplished by relating each PSG function to the control of its corresponding register. The function of creating or programming a specific sound or sound effect logically follows the control sequence listed:

Operation	Registers	Function
Tone Generator Control	R0--R5	Program tone periods.
Noise Generator Control	R6	Program noise period.
Mixer Control	R7	Enable tone and/or noise on selected channels.
Amplitude Control	~~R10--R12~~ R8 - R10	Select "fixed" or "envelope-variable" amplitudes.
Envelope Generator Control	~~R13--R15~~ R11 - R13	Program envelope period and select envelope pattern

Tone Generator Control
(Registers R0, R1, R2, R3, R4, R5)

The frequency of each square wave generated by the three Tone Generators (one each for Channels A, B, and C) is obtained in the PSG by first counting down the input clock by 16, then by further counting down the result by the programmed 12-bit Tone Period value. Each 12-bit value is obtained in the PSG by combining the contents of the relative Coarse and Fine Tune registers, as illustrated in the following:

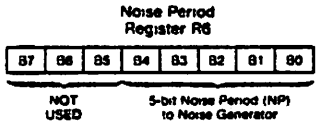
Coarse Tune Register	Channel	Fine Tune Register
R1	A	R0
R3	B	R2
R5	C	R4



12-bit Tone Period (TP) to Tone Generator

Noise Generator Control
(Register R6)

The frequency of the noise source is obtained in the PSG by first counting down the input clock by 16, then by further counting down the result by the programmed 5-bit Noise Period value. This 5-bit value consists of the lower 5 bits (B4--B0) of register R6, as illustrated in the following:



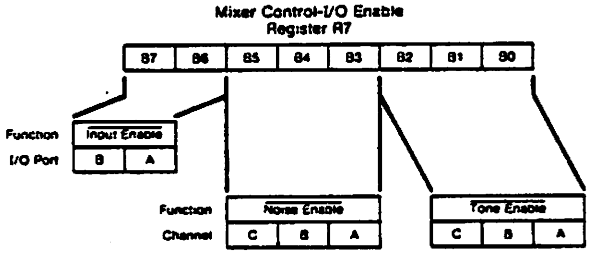
Mixer Control-I/O Enable
(Register R7)

Register 7 is a multi-function $\overline{\text{Enable}}$ register which controls the three Noise/Tone Mixers and the two general purpose I/O Ports.

The Mixers, as previously described, combine the noise and tone frequencies for each of the three channels. The determination of combining neither/either/both noise and tone frequencies on each channel is made by the state of bits B5--B0 of R7.

The direction (input or output) of the two general purpose I/O Ports (IOA and IOB) is determined by the state of bits B7 and B6 of R7.

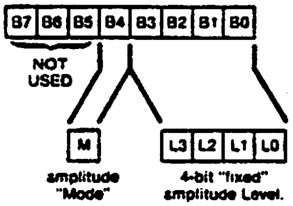
These functions are illustrated in the following:



Amplitude Control
(Registers ~~R10, R11, R12~~) Reg 8, 9, 10

The amplitudes of the signals generated by each of the three D/A Converters (one each for Channels A, B, and C) is determined by the contents of the lower 5 bits (B4--B0) of registers ~~R10, R11~~, and ~~R12~~ as illustrated in the following: R8, R9 R10

Amplitude Control Register #	Channel
R8 ~~R10~~	A
R9 ~~R11~~	B
R10 ~~R12~~	C

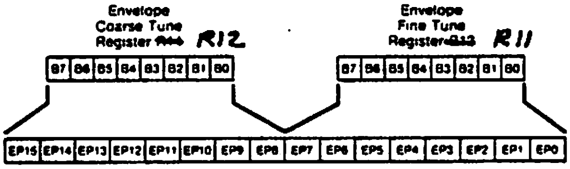


Envelope Generator Control
(Registers ~~R13, R14, R15~~) Reg 11, 12, 13

To accomplish the generation of fairly complex envelope patterns, two independent methods of control are provided in the PSG. first, it is possible to vary the frequency of the envelope using registers ~~R13~~ R11 and ~~R14~~ R12 and second, the relative shape and cycle pattern of the envelope can be varied using register R15. R13 The following paragraphs explain the details of the envelope control functions, describing first the envelope period control and then the envelope shape/cycle control.

ENVELOPE PERIOD CONTROL (Registers ~~R13, R14~~) R11, R12

The frequency of the envelope is obtained in the PSG by first counting down the input clock by 256, then by further counting down the result by the programmed 16-bit Envelope Period value. This 16-bit value is obtained in the PSG by combining the contents of the Envelope Coarse and Fine Tune registers, as illustrated in the following:

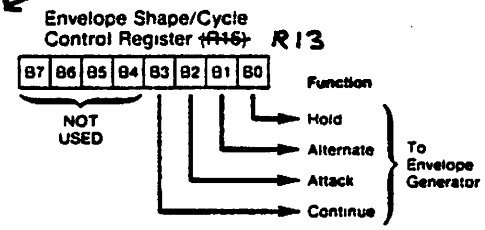

Envelope Coarse Tune Register ~~R14~~ R12
Envelope Fine Tune Register ~~R13~~ R11
B7
B6
B5
B4
B3
B2
B1
B0
EP15
EP14
EP13
EP12
EP11
EP10
EP9
EP8
EP7
EP6
EP5
EP4
EP3
EP2
EP1
EP0

16-bit Envelope Period (EP) to Envelope Generator

ENVELOPE SHAPE/CYCLE CONTROL (Register ~~R15~~) R 13

The Envelope Generator further counts down the envelope frequency by 16, producing a 16-state per cycle envelope pattern as defined by its 4-bit counter output, E3 E2 E1 E0. The particular shape and cycle pattern of any desired envelope is accomplished by controlling the count pattern (count up/count down) of the 4-bit counter and by defining a single-cycle or repeat-cycle pattern.

This envelope shape/cycle control is contained in the lower 4 bits (B3--B0) of register R15. R13 Each of these 4 bits controls a function in the envelope generator, as illustrated in the following:



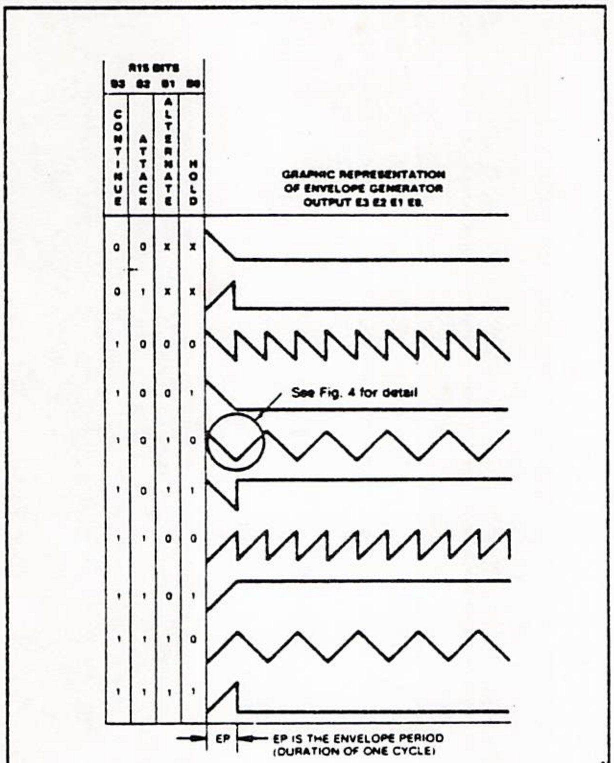


Fig. 3 ENVELOPE SHAPE/CYCLE OPERATION

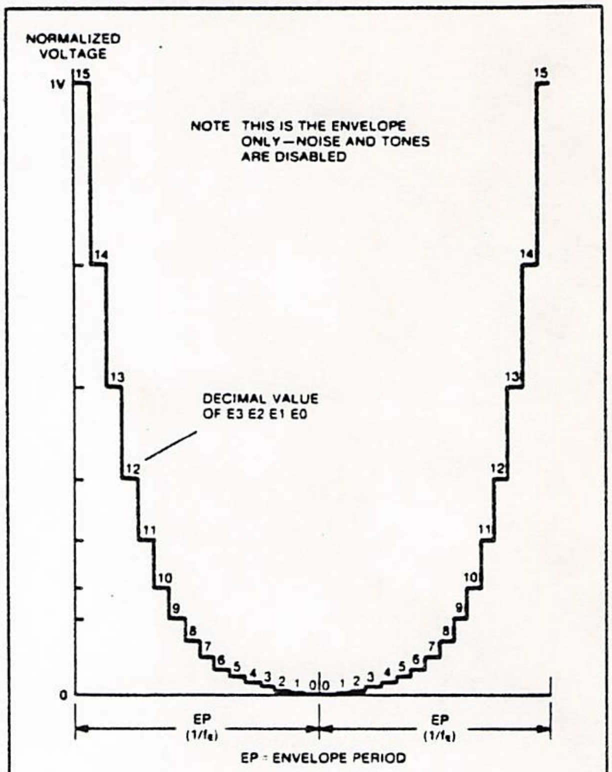


Fig. 5 D/A CONVERTER OUTPUT

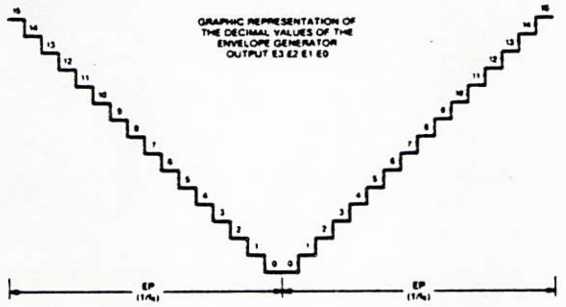


Fig. 4 DETAIL OF TWO CYCLES OF Fig. 3 (ref. waveform "1010" in Fig. 3)

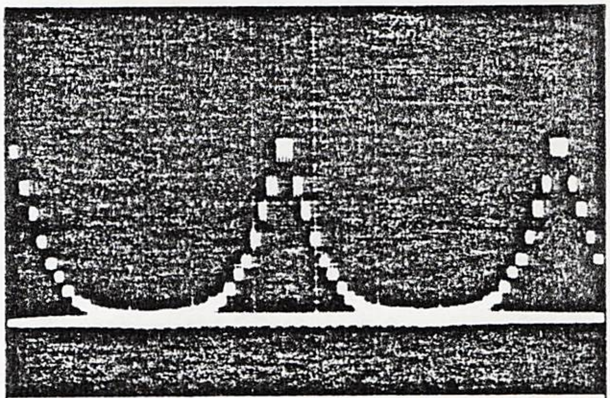


Fig. 6 SINGLE TONE WITH ENVELOPE SHAPE/CYCLE PATTERN 1010

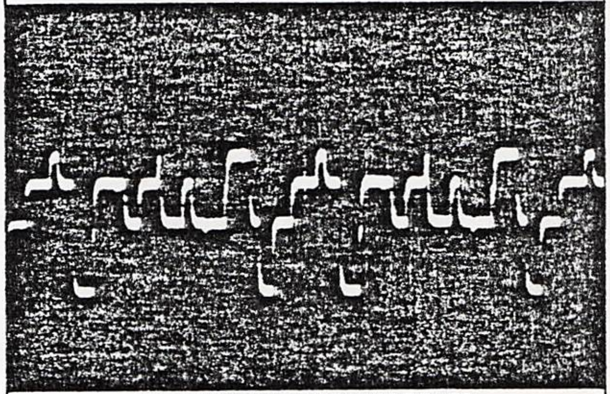


Fig. 7 MIXTURE OF THREE TONES WITH FIXED AMPLITUDES

I/O Port Data Store (Registers ~~R16, R17~~ R14, R15

Registers ~~R16~~ R14 and ~~R17~~ R15 function as intermediate data storage registers between the PSG/CPU data bus (DA0—DA7) and the two I/O ports (IOA7—IOA0 and IOB7—IOB0). Both ports are available in the AY-3-8910; only I/O Port A is available in the AY-3-8912. Using registers ~~R16~~ R14 and ~~R17~~ R15 for the transfer of I/O data has no effect at all on sound generation.

D/A Converter Operation

Since the primary use of the PSG is to produce sound for the highly imperfect amplitude detection mechanism of the human ear, the D/A conversion is performed in logarithmic steps with a normalized voltage range of from 0 to 1 Volt. The specific amplitude control of each of the three D/A Converters is accomplished by the three sets of 4-bit outputs of the Amplitude Control block, while the Mixer outputs provide the base signal frequency (Noise and/or Tone).

PLEASE NOTE:
===========

This is the Second Edition of the Manual and the content is the final version. The manual requires some minor editing before being put into its final spiral-bound format with half-size pages and small print. References such as "Preliminary Manual" will be edited out in the final format. Hence, we ask you to please ignore all references to this being a Preliminary Manual. On receipt of your warranty you will not be eligible to the manual in it's final format. Although this unformatted manual is costly for us to produce it is to your advantage as with the larger print it is easier to read and the loose-leaf pages can be easily xeroxed.